# DISEÑO, ENTRENAMIENTO E IMPLEMENTACIÓN DE AGENTES INTELIGENTES MEDIANTE APRENDIZAJE POR REFUERZO Y DEEP Q-NETWORKS

**JULIÁN ORLANDO RODRÍGUEZ VILLAMIZAR**

**DANIEL FELIPE RUEDA MARIÑO**

**UNIVERSIDAD INDUSTRIAL DE SANTANDER**

**FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS**

**ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**BUCARAMANGA**

**2024**

# DISEÑO, ENTRENAMIENTO E IMPLEMENTACIÓN DE AGENTES INTELIGENTES MEDIANTE APRENDIZAJE POR REFUERZO Y DEEP Q-NETWORKS

**JULIÁN ORLANDO RODRÍGUEZ VILLAMIZAR**

**DANIEL FELIPE RUEDA MARIÑO**

**Trabajo de investigación presentado para optar al título de Ingeniero de Sistemas**

**Director:**

**DAVID EDMUNDO ROMO BUCHELI**

**Ph.D en Ingeniería Eléctrica**

**Codirector:**

**SERGIO CASTILLO CASTELBLANCO**

**Ph.D en Ingeniería de Sistemas Telemáticos**

**UNIVERSIDAD INDUSTRIAL DE SANTANDER**

**FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS**

**ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**BUCARAMANGA**

**2024**

**Dedicatoria**

A mi madre Lissette, que se sacrificó y siempre cubrió a totalidad mis gastos para que me enfocara sólo en estudiar, a mis nonos Cecilia y Pedro, de los cuales siempre tuve el apoyo desde casa; Los cuales vieron en mí alguien capaz para afrontar todo esto. A mi primo Carlos, con el que crecí, siendo como un hermano para mí. A mi tío Camilo, que siempre me ha tenido un gran aprecio. A mi tía Claudia que ha sido el soporte de mi abuela y ha colaborado en el hogar en estos últimos años. A mi tía Nelly que siempre me ha dado su apoyo incondicional y ha estado pendiente de mí. A mi tía Berta que siempre me acompañó y fué un soporte importante. A mi tía Silvia que siempre me recuerda con cariño desde la distancia y en sus visitas me ha brindado maravillosos recuerdos. A mis múltiples relaciones fallidas durante mis tiempos universitarios. A mis amigos que han estado yendo y viniendo durante estos últimos años, a mi compañero de tesis que me acompañó desde el colegio por esta travesía. A los ratos amenos que he compartido con mi amigo “Viejo Tara”. A “mi mascota” fallecida Mateo que me acompaño durante la mayoría de mi carrera, y a la gatica “missi”, la “nueva mascota” que me da su compañía.

**Julían Rodríguez Villamizar**

**Dedicatoria**

Le dedico este Trabajo de Grado a mis padres y a mi hermano, las personas más valiosas que hay en mi vida, que siempre me han apoyado y creído en mí. A mi gatica Midna Alejandra y a mi tía Martha, y a mi tiíta Adriana y mi nonita Judith que ya no están con nosotros. A los Profesores que me han educado y me han tenido paciencia desde la escuela Primaria y sobre todo a la Universidad Industrial de Santander por aceptarme como estudiante y abrirme las puertas del campus en busca de un mejor futuro Profesional, muchas gracias a todos!!!!

**Daniel Felipe Rueda Mariño**

**Agradecimientos**

Expresamos en conjunto nuestro agradecimiento hacia:
A nuestro Director de tésis David Edmundo Romo Bucheli que nos tuvo paciencia y que nos acompañó en la mayoría de este camino como Director, que más que sólo dirigirnos, siempre estuvo pendiente de nosotros, siendo muy pro-activo con cada uno de los detalles sobre el proyecto. A la Universidad Industrial de Santander que nos acogió desde el primer instante que llegamos a ser parte de esta comunidad. A todos los profesores que tuvimos durante la carrera. A las personas del grupo de investigación BIVL2AB que a pesar de no pertenecer a este nos acogieron y siempre estuvieron dispuestos a ayudarnos.

**CONTENIDO**

**pág.**

**INTRODUCCIÓN** . . . . . **17**

**1. DESCRIPCIÓN DEL PROYECTO** . . . . . **18**

1.1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA . . . . . 18

1.2. OBJETIVOS . . . . . 20

1.2.1. Objetivo General. . . . . 20

1.2.2. Objetivos Específicos. . . . . 20

1.3. ALCANCE . . . . . 20

1.4. ROLES DE LOS MIEMBROS DEL EQUIPO DE TRABAJO . . . . . 21

**2. MARCO DE REFERENCIA Y HERRAMIENTAS** . . . . . **22**

2.1. MARCO TEÓRICO . . . . . 22

2.1.1. INTELIGENCIA ARTIFICIAL . . . . . 22

2.1.2. PROCESOS DE DECISIÓN DE MARKOV . . . . . 24

2.1.3. APRENDIZAJE POR REFUERZO . . . . . 25

2.1.4. Q-LEARNING . . . . . 27

2.1.5. DEEP LEARNING . . . . . 51

2.1.6. DEEP Q-NETWORKS . . . . . 52

2.2. ESTADO DEL ARTE . . . . . 54

2.2.1. DEEPMIND Y ALPHAGO . . . . . 54

2.2.2. JUEGOS DE ATARI 2600 . . . . . 58

2.2.3. APRENDIZAJE POR REFUERZO EN IMAGENES . . . . . 59

2.3. HERRAMIENTAS . . . . . 61

2.3.1. LIBRERÍAS PYTHON . . . . . 61

2.3.2. GOOGLE COLABORATORY . . . . . 63
2.3.3. PYCHARM . . . . . 63
2.3.4. OPENAI GYM . . . . . 63
2.3.5. HARDWARE . . . . . 66

**3. METODOLOGÍA** . . . . . **67**

**4. PÉNDULO INVERTIDO** . . . . . **68**
4.1. CONSTRUCCIÓN DE LOS MODELOS . . . . . 68
4.1.1. MODELO DE REFERENCIA . . . . . 68
4.1.2. MODELO PROPUESTO . . . . . 70
4.2. ENTRENAMIENTO DE LOS MODELOS . . . . . 71
4.2.1. SELECCIONAR ACCIÓN . . . . . 72
4.2.2. DISEÑO DE LA MEMORIA DEL AGENTE . . . . . 72
4.2.3. RECOMPENSA CON DESCUENTO . . . . . 74
4.2.4. CÁLCULO DE LA PÉRDIDA . . . . . 75
4.2.5. FUNCIÓN DE ENTRENAMIENTO . . . . . 76
4.2.6. ALGORITMO GENERAL . . . . . 77
4.3. COMPARAR DESEMPEÑO DE LOS AGENTES . . . . . 82

**5. PONG** . . . . . **86**
5.1. CONSTRUCCIÓN DE LOS MODELOS . . . . . 86
5.1.1. MODELO DE REFERENCIA. . . . . 86
5.1.2. MODELO PROPUESTO. . . . . 91
5.2. ENTRENAMIENTO DE LOS MODELOS . . . . . 93
5.2.1. SELECCIONAR ACCIÓN . . . . . 93
5.2.2. DISEÑO DE LA MEMORIA DEL AGENTE . . . . . 93
5.2.3. FUNCIÓN DE ENTRENAMIENTO . . . . . 93
5.2.4. CÁLCULO DE LA PÉRDIDA . . . . . 93

5.2.5. RECOMPENSA CON DESCUENTO . . . . . 93
5.2.6. CONFIGURACIÓN PREVIA AL ENTRENAMIENTO . . . . . 95
5.2.7. ALGORITMO GENERAL . . . . . 96
5.3. COMPARAR DESEMPEÑO DE LOS AGENTES . . . . . 98

**6. DEEP RECURRENT ATTENTION MODEL PARA IDENTIFICAR REGIONES DE INTERÉS CLÍNICO EN IMÁGENES HISTOLÓGICAS . . . . . 104**
6.1. MATERIALES . . . . . 104
6.2. HISTODRAM . . . . . 107
6.2.1. ENTRENAMIENTO DEL MODELO . . . . . 110
6.3. EVALUACIÓN DE MAPAS DE DENSIDAD DE PROBABILIDAD . . . . . 112
6.4. OBSERVACIONES FINALES . . . . . 117

**7. CONCLUSIONES Y PERSPECTIVAS . . . . . 118**

**BIBLIOGRAFÍA . . . . . 119**

**ANEXOS . . . . . 128**

## LISTA DE FIGURAS

**pág.**


Figura 1. Procesos de Decisión de Markov . . . . . 24
Figura 2. Estructura básica del Apredizaje por Refuerzo . . . . . 26
Figura 3. Q-Learning . . . . . 27
Figura 4. Recompensa Acumulada con Descuento . . . . . 28
Figura 5. Método $\varepsilon$ -Greedy . . . . . 31
Figura 6. Ecuaciones de Bellman . . . . . 32
Figura 7. Problema del Lago Congelado . . . . . 34
Figura 8. Ruta Episodio 1 . . . . . 38
Figura 9. Ruta Episodio 2 . . . . . 40
Figura 10. Ruta Episodio 3 . . . . . 42
Figura 11. Ruta Episodio 4 . . . . . 44
Figura 12. Ruta Episodio 5 . . . . . 46
Figura 13. Ruta Episodio 6 . . . . . 48
Figura 14. Ruta Episodio 7 . . . . . 50
Figura 15. Deep Learning . . . . . 51
Figura 16. Deep Q-Network . . . . . 52
Figura 17. Juego de Mesa Go . . . . . 55
Figura 18. Árbol de Búsqueda Montecarlo . . . . . 56
Figura 19. CartPole-v0 . . . . . 64
Figura 20. Pong-v0 . . . . . 65
Figura 21. Modelo de referencia Péndulo Invertido . . . . . 69
Figura 22. Espacio de acción Péndulo Invertido . . . . . 70
Figura 23. Modelo propuesto Péndulo Invertido . . . . . 71

Figura 24. Función choose_action Péndulo Invertido . . . . . 72
Figura 25. Memoria del agente Péndulo Invertido . . . . . 73
Figura 26. Recompensa con descuento Péndulo Invertido . . . . . 74
Figura 27. Cálculo de la pérdida Péndulo Invertido . . . . . 75
Figura 28. Función de entrenamiento Péndulo Invertido . . . . . 76
Figura 29. Entrenamiento de los Modelos Péndulo Invertido . . . . . 78
Figura 30. Algoritmo general Péndulo Invertido . . . . . 79
Figura 31. Comparación Learning Rate . . . . . 80
Figura 32. Resultados Péndulo Invertido modelo de referencia . . . . . 82
Figura 33. Función generar video Péndulo Invertido . . . . . 83
Figura 34. Resultados Péndulo Invertido modelo propuesto . . . . . 84

Figura 35. Modelo de referencia Pong . . . . . 86
Figura 36. Espacio de acción Pong . . . . . 87
Figura 37. Capa convolucional padding y kernel . . . . . 88
Figura 38. Producto Kernel Input Capa convolucional . . . . . 89
Figura 39. Flattening . . . . . 91
Figura 40. Modelo propuesto Pong . . . . . 92
Figura 41. Recompensa con descuento Pong . . . . . 94
Figura 42. Configuración previa Pong . . . . . 95
Figura 43. Algoritmo general Pong . . . . . 97
Figura 44. Resultados Pong modelo de referencia . . . . . 99
Figura 45. Resultados Pong modelo propuesto primer intervalo . . . . . 101
Figura 46. Resultados Pong modelo propuesto último intervalo . . . . . 102
Figura 47. Comparación de Puntajes entre modelos (Pong) . . . . . 103

Figura 48. Representación piramidal de las imágenes histológicas. . . . . 105
Figura 49. Deep Recurrent Attention Model . . . . . 108
Figura 50. Resultados DRAM . . . . . 112

Figura 51. Resultados cualitativos para los mapas de densidad de la evaluación115
Figura 52. Comparación de Porcentaje de área de Cáncer en tejido . . . . . 116

## LISTA DE CUADROS

**pág.**


Cuadro 1. Registros Episodio 1 . . . . . 36
Cuadro 2. Q-Table después del Episodio 1 . . . . . 37
Cuadro 3. Registros Episodio 2 . . . . . 39
Cuadro 4. Q-Table después del Episodio 2 . . . . . 40
Cuadro 5. Registros Episodio 3 . . . . . 41
Cuadro 6. Q-Table después del Episodio 3 . . . . . 41
Cuadro 7. Registros Episodio 4 . . . . . 43
Cuadro 8. Q-Table después del Episodio 4 . . . . . 44
Cuadro 9. Registros Episodio 5 . . . . . 45
Cuadro 10. Q-Table después del Episodio 5 . . . . . 46
Cuadro 11. Registros Episodio 6 . . . . . 47
Cuadro 12. Q-Table después del Episodio 6 . . . . . 48
Cuadro 13. Registros Episodio 7 . . . . . 49
Cuadro 14. Q-Table después del Episodio 7 . . . . . 50

Cuadro 15. ISUP Grade y Gleason Score . . . . . 106
Cuadro 16. DRAM Implementation experiments. . . . . 111
Cuadro 17. Promedio y Desviación Estándar de Métricas . . . . . 114

## RESUMEN

**TÍTULO:** DISEÑO, ENTRENAMIENTO E IMPLEMENTACIÓN DE AGENTES INTELIGENTES MEDIANTE APRENDIZAJE POR REFUERZO Y DEEP Q-NETWORKS [*]

**AUTOR:** DANIEL FELIPE RUEDA MARIÑO
JULIÁN ORLANDO RODRÍGUEZ VILLAMIZAR [**]

**PALABRAS CLAVE:** APRENDIZAJE POR REFUERZO, Q-NETWORKS, Q-LEARNING, APRENDIZAJE AUTOMÁTICO, AGENTE INTELIGENTE.

**DESCRIPCIÓN:** Los agentes inteligentes entrenados mediante modelos de inteligencia artificial son capaces de comportarse de una u otra manera dependiendo de la situación (se busca que estos actúen "inteligentemente", como su nombre lo indica). Son utilizados para realizar de manera más eficiente diversas funciones de tal forma que se desempeñen en las mismas mejor que como lo haría una persona.

Para ello es necesario entrenar a dichos agentes de tal manera que aprendan a actuar adecuadamente acorde a la situación dependiendo de la labor que tengan que realizar. En el ámbito de la inteligencia artificial se puede recurrir a una gran variedad de paradigmas para entrenar modelos, para el desarrollo del trabajo se utilizará el Aprendizaje por Refuerzo con el propósito de entrenar adecuadamente agentes inteligentes capaces de aprender de sus errores y tomar mejores decisiones a futuro, con el objetivo de adquirir la mayor recompensa posible, la cual se obtiene cuando el agente se desempeña adecuadamente en su labor.


---

[*] Trabajo de investigación

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: David Edmundo Romo Bucheli, Ph.D en Ingeniería Eléctrica.

## ABSTRACT

**TITLE:** DESIGN, TRAINING, AND IMPLEMENTATION OF INTELLIGENT AGENTS THROUGH REINFORCEMENT LEARNING AND DEEP Q-NETWORKS. [*]

**AUTHOR:**

**AUTOR:** DANIEL FELIPE RUEDA MARIÑO
JULIÁN ORLANDO RODRÍGUEZ VILLAMIZAR [**]

**KEYWORDS:** REINFORCEMENT LEARNING, Q-NETWORKS, Q-LEARNING, MACHINE LEARNING, INTELLIGENT AGENT.

**DESCRIPTION:** Intelligent agents trained using artificial intelligence models can behave in one way or another depending on the situation (the goal for these agents is to act "intelligently", as their name suggests). They are used to perform more efficiently a lot of actions better than a person would do. For that reason, it is necessary to train intelligent agents in such a way that they learn to act properly according to the situation depending on the work they have to do. In the field of AI a wide variety of paradigms can be used to train models, Reinforcement Learning will be used for the development of this work with the purpose of adequately train intelligent agents capable of learning from their mistakes to make better future decisions, with the aim of acquiring the highest possible reward, which is obtained when the agent performs adequately in its work.


---

[*] Research work

[**] Faculty of Physical-Mechanical Engineering. School of Systems and Computer Engineering. Advisor: David Edmundo Romo Bucheli, Ph.D in Electrical Engineering

## ESTRUCTURA DEL PROYECTO

El trabajo se llevará a cabo con base en los siguientes capítulos que servirán para explicar la temática y el desarrollo del Proyecto junto a las conclusiones que se obtendrán tras ello.

**Capítulo 1.** Descripción del proyecto: Consiste en el planteamiento y la justificación del problema que se busca solucionar mediante el desarrollo del proyecto propuesto, junto a los objetivos que se buscan alcanzar. También se hace referencia a los alcances del proyecto, junto a la descripción de los roles que cada miembro del equipo de trabajo realizó.

**Capítulo 2.** Marco teórico y herramientas: Se explican los fundamentos teóricos del Aprendizaje por Refuerzo, el estado del arte y las herramientas que serán necesarias para el adecuado desarrollo del proyecto.

**Capítulo 3.** Metodología para el diseño y la implementación de Deep Q-Networks: Se hace referencia al desarrollo en espiral y a la metodología CRISP-DM. También se definen las fases que compondrán el desarrollo y la implementación de los Agentes Inteligentes que se buscan diseñar.

**Capítulo 4.** Péndulo Invertido: Se exponen los algoritmos utilizados para entrenar un Agente Inteligente capaz de superar el Problema del Péndulo Invertido mediante el Aprendizaje por Refuerzo, las Redes Neuronales usadas para el entrenamiento de los Agentes y los resultados obtenidos.

**Capítulo 5.** Pong: Se exponen los algoritmos utilizados, las Redes Neuronales usadas para el entrenamiento y los resultados obtenidos tras implementar Aprendizaje por Refuerzo para que un Agente pueda superar el entorno del videojuego de Atari 2600 Pong.

**Capítulo 6.** Deep Recurrent Attention Model para identificar regiones de interés clínico en imágenes histológicas: Se diseñan los agentes inteligentes utilizados para identificar si una biopsia de una próstata tiene cáncer o no, y se entrenan mediante la Deep Recurrent Attention Model. Este modelo aprende a centrar su atención mayoritariamente en las áreas de mayor relevancia clínica para que el problema de clasificación se resuelva lo más fácilmente posible.

**Capítulo 7.** Conclusiones y perspectivas

# INTRODUCCIÓN

El Aprendizaje por Refuerzo es un paradigma de Machine Learning utilizado para diseñar Inteligencias Artificiales que aprenden a comportarse de manera adecuada a partir de sus errores cometidos con anterioridad. A diferencia del Aprendizaje Supervisado y No-Supervisado en este caso el modelo no se entrena con un dataset ya definido, por el contrario, este se va creando a medida que el agente va interactuando con su entorno.

El agente inteligente que se va a entrenar mediante Aprendizaje por Refuerzo puede llevar a cabo una gran cantidad de acciones dependiendo del entorno en el que se desenvuelva, este va explorando dicho entorno de tal manera que va llevando a cabo acciones que alteran el estado de este y como consecuencia se van adquiriendo recompensas o penalizaciones dependiendo de si el agente actúa de manera adecuada o inadecuada.

Por lo que los agentes que se entrenan mediante Aprendizaje por Refuerzo van diseñando una política de comportamiento a seguir, de tal manera que se evita llevar ciertas acciones a cabo en determinada situación, y por el contrario se busca que la mayoría de las veces cuando se requiera, se seleccione la acción que más recompensa acumulada le ha generado al agente en el pasado.

De esa manera el Aprendizaje por Refuerzo sirve para que los agentes inteligentes aprendan a comportarse acorde a la situación. Se recurrirá a Deep Q-Networks para entrenar los modelos con los que se trabajará durante el desarrollo del trabajo. Los entornos en los que los agentes se desenvolverán serán otorgados por OpenAI Gym.

# 1. DESCRIPCIÓN DEL PROYECTO

## 1.1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

Actualmente la Inteligencia Artificial evoluciona de manera acelerada. Es de vital importancia recurrir a ésta debido a que sobrepasa las capacidades de procesamiento de información que poseen los seres humanos, por ende perfecciona procesos y permite que se lleven a cabo diversas actividades tal y como las realizaría un ser humano e incluso puede llegar a desempeñarse en las mismas mejor que éste.

Uno de los campos más estudiados en el ámbito de la Inteligencia Artificial es el del Machine Learning, recurrir a modelos de redes neuronales capaces de entrenarse para que al recibir determinadas entradas logren predecir de la manera más acertada posible la salida deseada como si de una función se tratara.[1]

Existe una gran variedad de paradigmas de Machine Learning que se tienen en cuenta para el diseño y el entrenamiento de los modelos de redes neuronales.[2] El Aprendizaje por Refuerzo es uno de estos, y la mayor diferencia que existe entre este paradigma y otros (Aprendizaje Supervisado y no-Supervisado), es que el modelo se entrena mediante un dataset que se va generando y actualizando constantemente.

Existen situaciones en las que no se conoce la totalidad de la información que se va a utilizar para entrenar un modelo de red neuronal. En estos casos es aconsejable

---

[1] BONACCORSO, Giuseppe. *Machine Learning Algorithms*. Jul. de 2017.

[2] (PARIKH Dhairya, 2018)

recurrir al Aprendizaje por Refuerzo, para que se vaya conociendo el entorno en el que se busca que el agente inteligente se desenvuelva adecuadamente, debido a que es necesario que éste lleve a cabo acciones en dicho entorno para adquirir recompensas y penalizaciones. De esa manera el agente va adquiriendo la información requerida para entrenarse de manera adecuada y de esa forma evitar las penalizaciones lo más que pueda.

Por ende, el desarrollo del Proyecto se centra en el Aprendizaje por Refuerzo, resulta indispensable recurrir a éste para cuando no se conoce la totalidad del dataset que se utiliza de referencia para entrenar un modelo de red neuronal. Cabe aclarar que además de trabajar con los entornos proporcionados por openAI Gym,[3] el Aprendizaje por Refuerzo también puede aplicarse en la vida real, en el ámbito de los sistemas de control,[4] los vehículos automatizados,[5] entre muchas otras aplicaciones.

---

[3] (OPENAI GYM. Gym [En línea]. [Consultado: 16 de Mayo del 2021]. Disponible en: https://gym.openai.com/docs/)

[4] HAYDARI, Ammar e YILMAZ, Yasin. "Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey". En: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2022), págs. 11-32. DOI: 10.1109/TITS.2020.3008612.

[5] B, Ravi, Kiran., Ibrahim, Sobh., Victor, Talpaert., Patrick, Mannion., Ahmad, A., Al, Sallab., Senthil, Yogamani., Patrick, Pérez. (2021). Deep Reinforcement Learning for Autonomous Driving: A Survey. IEEE Transactions on Intelligent Transportation Systems, 1-18. doi: 10.1109/TITS.2021.3054625

## 1.2. OBJETIVOS

**1.2.1. Objetivo General.** Desarrollar agentes inteligentes usando entornos openAI Gym y Deep Q-Networks, mediante Aprendizaje por Refuerzo para crear políticas de comportamiento que mejoren la eficiencia de los agentes al momento de tomar decisiones.

### 1.2.2. Objetivos Específicos.

1) Examinar el apartado teórico del Aprendizaje por Refuerzo y su respectiva implementación matemática.
2) Diseñar y entrenar un agente de tal manera que logre mantener el equilibrio del sistema en el problema del Péndulo Invertido y llevar a cabo la respectiva evaluación de resultados comparando las recompensas adquiridas con base en realizar el entrenamiento mediante Q-Networks diferentes.
3) Diseñar y entrenar un agente capaz de jugar y aprender el juego de Atari 2600 Pong (basado en el juego de mesa Ping Pong), y llevar a cabo la respectiva evaluación de resultados comparando las recompensas adquiridas con base en realizar el entrenamiento mediante Deep Q-Networks diferentes.
4) Diseñar y entrenar un agente capaz de identificar regiones de interés clínico en imágenes histológicas utilizando Aprendizaje por Refuerzo y una Deep Recurrent Attention Model, partiendo de una tarea de clasificación débilmente supervisada.

## 1.3. ALCANCE

Los resultados obtenidos se ven limitados por la capacidad computacional de la máquina compartida del grupo de investigación HDSP y de Google Colaboratory. El trabajo es llevado a cabo por 2 estudiantes debido a lo extenso de la temática (ya que

son 3 aplicaciones del Aprendizaje por Refuerzo) y la complejidad de ésta. Trabajar en equipo resulta ser una ventaja debido a que de esa manera cada miembro de éste se encarga de un ámbito diferente (documentación, investigación, programación), de tal forma que cada uno se especializa en aquello que le resulta más sencillo de explicar y comprender, se tienen puntos de vista diferentes y por ende se comparten ideas que al debatirse y analizarse detenidamente sirven para llegar a consensos y tomar decisiones, en busca de la adecuada realización del Proyecto.

## 1.4. ROLES DE LOS MIEMBROS DEL EQUIPO DE TRABAJO

-Daniel Rueda: Encargado de redactar la documentación necesaria para la adecuada exposición del tema del Trabajo de Grado de manera concisa, entendible y organizada.Además, supervisa la calidad de la documentación y coordinar en equipo, para asegurar la coherencia en la presentación del trabajo. Se encarga de la gestión de plazos y recursos relacionados con la documentación
-Julián Orlando: Encargado de la investigación y la adquisición de aquellos trabajos, libros y publicaciones científicas que se toman como referencia para la redacción del Trabajo de Grado. Además de encargarse de llevar a cabo y documentar los experimentos necesarios para el entrenamiento de los modelos.
-En conjunto: Comprender, adaptar y formular el código necesario para la adecuada implementación del Aprendizaje por Refuerzo en los 3 casos de estudio.

# 2. MARCO DE REFERENCIA Y HERRAMIENTAS

## 2.1. MARCO TEÓRICO

**2.1.1. INTELIGENCIA ARTIFICIAL** La Inteligencia Artificial se implementa en una gran cantidad de las empresas y organizaciones que existen actualmente,[6] la demanda en cuanto a la necesidad de recurrir a ésta va en aumento debido a lo rentable que resulta ser utilizar Inteligencia Artificial en procesos de negocios,[7]. ya que los modelos entrenados en dicho ámbito pueden llegar a comportarse como seres vivos capaces de razonar y de tomar las decisiones acertadas en el momento adecuado dependiendo del análisis de información que previamente se haya llevado a cabo y de la situación o problemática que se busca resolver.

La Inteligencia Artificial se diseña mediante algoritmos a los que recurre una máquina o agente, de tal forma que éste es capaz de simular llevar a cabo funciones cognitivas como las que ocurren en el cerebro humano en el ámbito de la biología, por ello se dice que las máquinas que se entrenan mediante Inteligencia Artificial son capaces de aprender.[8]

---

6 BERNARD MARR, Matt Ward. *How 50 Successful Companies Used AI and Machine Learning to Solve Problems*. Wiley, 2019.

7 EKKEHARDT ERNST, Rossana Merola y Daniel Samaan. “Economics of Artificial Intelligence: Implications for the Future of Work”. En: *IZA Journal of Labor Policy* 9.Jun (2019).

8 KOTELUK, Oliwia, *et al.* “How Do Machines Learn? Artificial Intelligence as a New Era in Medicine”. En: *Journal of Personalized Medicine* 11.1 (2021). DOI: 10.3390/jpm11010032.

Para implementarla se puede recurrir al Machine Learning,[9] mediante este método se suelen utilizar redes neuronales (Deep Learning)[10] para entrenar modelos y posteriormente comparar resultados y eficiencia. Existen diversos paradigmas los cuales se pueden tener en cuenta para dicho proceso de entrenamiento. El más común es el del Aprendizaje Supervisado, en el que el diseñador de la IA se ve en la necesidad de conocer en su totalidad una muestra significativa de las entradas y las respectivas salidas que se supone que deberían obtenerse mediante el modelo, con base en dicha información el modelo se entrena y posteriormente se utiliza para predecir los resultados esperados con la mayor precisión posible. [11]. En cambio, cuando se entrena un modelo mediante el Aprendizaje no-Supervisado únicamente se conocen las entradas al sistema y el modelo analiza la información, determina los patrones ocultos presentes en ésta y de esa manera se genera la salida.[12] También existe otro paradigma de Machine Learning de vital importancia, sobre todo si se desea entrenar un modelo con una cantidad indefinida y desconocida de datos, el Aprendizaje por Refuerzo. A diferencia de los paradigmas mencionados con anterioridad éste permite trabajar con datasets que se van generando continuamente, y se suele utilizar para diseñar políticas de comportamiento para agentes que interactúan constantemente con su entorno, recibiendo bonificaciones y penalizaciones por ello. [11]

---

[9] ZHANG, Xian-Da. "Machine Learning". En: *A Matrix Algebra Approach to Artificial Intelligence*. Singapore: Springer Singapore, 2020, págs. 223-440. DOI: 10.1007/978-981-15-2770-8_6.

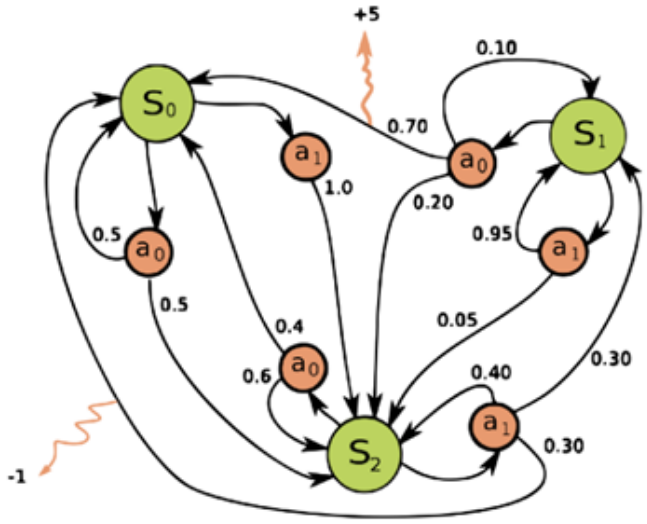
[10] GOODFELLOW, Ian; BENGIO, Yoshua y COURVILLE, Aaron. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[11] RAVICHANDIRAN, Sudharsan. *Hands-On Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow*. eng. Birmingham: Packt Publishing, Limited, 2018.

[12] CELEBI, M. Emre y AYDIN, Kemal. *Unsupervised Learning Algorithms*. eng. Cham: Springer International Publishing AG, 2016.

**2.1.2. PROCESOS DE DECISIÓN DE MARKOV** El Aprendizaje por Refuerzo es aplicado en sistemas con elementos y lógica similar a la descrita en los Procesos de Decisión de Markov.[13] Debido a que en ambos casos se trabaja con un conjunto de estados los cuales se pueden alcanzar tras llevar a cabo determinadas acciones en ciertos estados anteriores, teniendo en cuenta que siempre existe un estado inicial.[14]

**Figura 1.** Procesos de Decisión de Markov




**Fuente:** waldo ALVAREZ. *Example of a Markov Decision Process*. URL: https://commons.wikimedia.org/wiki/File:Markov_Decision_Process.svg

[13] OTTERLO, Martijn y WIERING, Marco. "Reinforcement Learning and Markov Decision Processes". En: *Reinforcement Learning: State of the Art* (ene. de 2012), págs. 3-42. DOI: 10.1007/978-3-642-27645-3_1.

[14] XIA, Li. "Mean–variance optimization of discrete time discounted Markov decision processes". eng. En: *Automatica (Oxford)* 88 (2018), págs. 76-82.

Tal y como se puede ver en la Figura 1, en los Procesos de Decisión de Markov existen estados S (los círculos verdes, teniendo en cuenta que el estado inicial se identifica por el S0), para ir de un estado S a otro es necesario llevar a cabo acciones "a" (los círculos anaranjados), y después de realizar una acción se tiene en cuenta una distribución de probabilidad para alcanzar uno u otro estado (en la Figura 1 esto se representa mediante las flechas que salen de cada acción que se puede realizar, éstas apuntan hacia los estados que probablemente se alcancen por la acción llevada a cabo aunque solo se llegará a uno de éstos, y la selección del estado que se alcanza depende de las probabilidades de selección ligadas a cada flecha). Tras llevar a cabo ciertas acciones en determinados estados se puede adquirir una recompensa o una penalización por ello (en la Figura 1 se representan por aquellos valores enteros positivos o negativos que se señalizan mediante una flecha anaranjada que a su vez surge de la transición existente de un estado a otro tras llevar a cabo la acción que genera su respectiva penalización o bonificación acorde a la situación). Dependiendo de las recompensas adquiridas se van actualizando las distribuciones de probabilidad existentes para cada dupla Estado-Acción y de esa manera se va generando una política de comportamiento, de tal forma que el estado más probable a alcanzar tras llevar a cabo una acción es aquel que permite que en un futuro se obtenga la mayor recompensa posible. [14]

**2.1.3. APRENDIZAJE POR REFUERZO** El Aprendizaje por Refuerzo es utilizado en los casos en los que un agente inteligente se entrena con base en la información que va procesando del entorno que analiza y aprende de sus errores. El nombre de este paradigma de Machine Learning proviene de la semejanza que existe entre el proceso de entrenamiento llevado a cabo por los agentes inteligentes que se entrenan mediante Aprendizaje por Refuerzo y el mundo real, debido a que por ejemplo, un perro es recompensado con alimento cada vez que lleva a cabo cierto truco, aprende a ir al baño en el exterior y no dentro de la casa, etc. Cuando al

animal se le otorga una recompensa (un refuerzo), se verá motivado a ir adquiriendo cada vez más y para ello se comportará de una determinada manera dependiendo de lo que busque el entrenador o dueño de la mascota. [11]

De igual forma un agente que se entrena adecuadamente con base en el Aprendizaje por Refuerzo irá adquiriendo cada vez más recompensa a medida que vaya llevando a cabo acciones que afectan el estado del entorno en el que se desenvuelve, tiene que analizar el sistema en el que actúa y elegir una de las posibles acciones que puede llevar a cabo. Entre más se entrene el agente (adecuadamente, claro está) mejores son las decisiones que toma en busca de evitar penalizaciones y por el contrario ir adquiriendo cada vez más recompensa y experiencia.

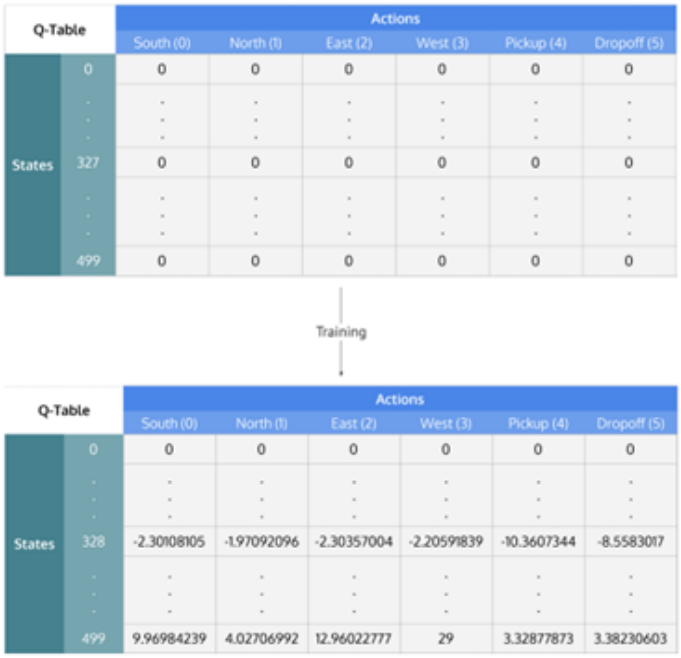
**Figura 2.** Estructura básica del Aprendizaje por Refuerzo.


**Fuente:** Elaboración propia

El ciclo de entrenamiento que se lleva a cabo para que los agentes vayan conociendo su entorno se define mediante la figura 2. El agente realiza una acción que afecta al entorno, y como consecuencia adquiere por ello una recompensa (nula, positiva o negativa acorde a la situación) y un nuevo estado del entorno, que el agente analizará para posteriormente llevar a cabo otra acción, y el ciclo se repite constantemente.

**2.1.4. Q-LEARNING** El Aprendizaje por Refuerzo se puede llegar a apoyar en el Q-Learning, un método de Aprendizaje Automático que consiste en ir calculando valores denominados Q-Values, que se van actualizando para cada dupla Estado-Acción a medida que el agente lleva a cabo acciones en el entorno en el que se desenvuelve (ver Figura 3).[15]

**Figura 3.** Q-Learning




**Fuente:** Satwik Kansal BRENDAN MARTIN. *Q-Learning Matrix Initialized and After Training*. URL: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

[15] WATKINS, Christopher J.C.H y DAYAN, Peter. "Technical Note: Q-Learning". eng. En: *Machine learning* 8.3 (1992), pág. 279.

Mediante los Q-Values se va generando la política de comportamiento a seguir por parte del agente, de tal manera que éstos en un primer momento adquieren valores aleatorios y a medida que se van consiguiendo recompensas y penalizaciones se van actualizando.

**RECOMPENSA ACUMULADA Y CON DESCUENTO** Después de que el agente lleva a cabo una acción en el estado del entorno en el que se encuentra, obtiene una recompensa positiva, nula o negativa dependiendo de la situación. Las recompensas adquiridas por el agente son indispensables para ir actualizando el valor de los Q-Values para cada estado del entorno, el problema radica en el hecho de que es necesario ir acumulando la recompensa debido a que el objetivo de un agente inteligente entrenado mediante Aprendizaje por Refuerzo es alcanzar la mayor recompensa posible, teniendo en cuenta que la suma de todas las recompensas no debe tender al infinito para simplificar el proceso de actualización de los Q-Values.
Para que el valor de la recompensa acumulada converja fácilmente se recurre al factor de descuento, por ende, al momento de actualizar un Q-Value se tiene en cuenta el parámetro denominado Recompensa Acumulada con Descuento, el cual se calcula mediante la fórmula expuesta en la Figura 4.

**Figura 4.** Recompensa Acumulada con Descuento.

$$R_t = \sum_{k=0}^{t-1} \gamma^k r_{k+1} \tag{1}$$

**Fuente:** Deep Reinforcement Learning: An Overview. (Li, 2018).

Los parámetros que se tienen en cuenta al momento de aplicar la fórmula son los siguientes:

- $r_{k+1}$:Recompensa inmediata y sin descuento que se obtiene tras llevarse a cabo una acción en el estado del entorno en el que se encuentra el agente

durante el timestep k+1.

- $R_t$:Recompensa acumulada y con descuento que se obtiene en el timestep t, después que el agente recibe una recompensa inmediata y sin descuento.

- $\sum_{k=0}^{t-1}$:La sumatoria en cuestión se tiene en cuenta para ir acumulando las recompensas adquiridas por el agente. k va aumentando a medida que el agente va recibiendo nuevas recompensas, hasta que alcanza el valor de t-1.

- $\gamma$: Hace referencia al factor de descuento, indispensable para que la recompensa acumulada converja y no tienda al infinito. El factor de descuento se encuentra entre 0 y 1, de tal manera que si éste adquiere un valor cercano a cero, las recompensas que más se tendrían en cuenta serían las del comienzo del proceso de entrenamiento. Por el contrario, si el factor de descuento es cercano a 1, las recompensas más valiosas son las cercanas al fin del episodio, de tal manera que el agente se ve en la necesidad de ir buscando mayores recompensas tras recibir las que ha obtenido para ir aumentando de manera significativa el valor que adquiere Rt a lo largo del episodio. En el ámbito del Aprendizaje por Refuerzo se suele trabajar con $\gamma$ cercano a 1, para que las recompensas más significativas sean aquellas que se han alcanzado en un estado avanzado del sistema, de tal manera que la recompensa se vaya maximizando a medida que el agente va aprendiendo a desempeñarse mejor en su labor, y porque las mejores recompensas suelen estar al final del episodio (cuando se gana una partida, cuando se logra acceder al siguiente nivel de un videojuego, etc).[16]

---

[16] YOSHIDA, Naoto; UCHIBE, Eiji y DOYA, Kenji. “Reinforcement learning with state-dependent discount factor”. eng. En: *2013 IEEE Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. IEEE, 2013, págs. 1-6.

Por lo que la Recompensa Acumulada con Descuento que se obtiene tras haber llevado a cabo una acción "a" en un estado s sería la suma de todas las recompensas adquiridas hasta el momento (incluyendo las recompensas iguales a cero), cada una multiplicada por el factor de descuento elevado a un número entero que va aumentando en 1 y que en un primer momento tiene el valor de 0 (por ende, la primera recompensa no se multiplica por factor de descuento alguno).[17]

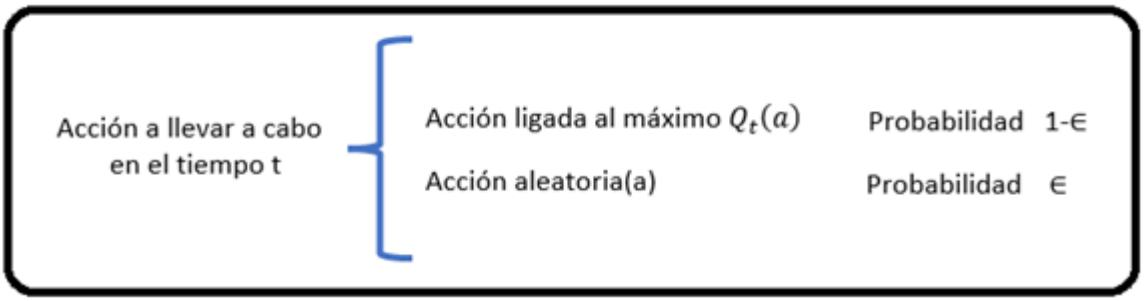
**Método $\varepsilon$ -Greedy:** Para obtener las recompensas y las penalizaciones necesarias para actualizar los Q-Values es necesario que el agente lleve a cabo acciones, para seleccionar la acción a realizar se puede recurrir al Método $\varepsilon$-Greedy (ver figura 5). Mediante la implementación de este método se busca que la mayoría de las veces el agente lleve a cabo las acciones ligadas a los Q-Values más altos para obtener la mayor recompensa posible. Existiendo siempre una muy pequeña posibilidad de que se seleccione una acción al azar, con el propósito de conocer nuevos estados del entorno y de esa manera adquirir nuevas recompensas a tener en cuenta durante el proceso de entrenamiento.[18]

---

[17] LI, Yuxi. "Deep Reinforcement Learning: An Overview". eng. En: *arXiv.org* (2017).

[18] DUTTA, Sayon. *Reinforcement Learning with TensorFlow: A Beginner's Guide to Designing Self-Learning Systems with TensorFlow and OpenAI Gym*. eng. Birmingham: Packt Publishing, Limited, 2018.

**Figura 5.** Método $\varepsilon$ -Greedy




**Fuente:** Elaboración propia

Las variables con las que se trabaja al aplicar el Método $\varepsilon$-Greedy expuesto en la Figura 5 son las siguientes:

- a:Acción que será llevada a cabo por el agente.
- $Q_t(a)$:Q-Value ligado a la acción "a" del estado en el que se encuentra el agente en el timestep t.
- $\varepsilon$:Valor presente en el intervalo [0, 1] que hace referencia a la probabilidad de que en el timestep t se opte por escoger una acción aleatoria en vez de elegir aquella ligada al mayor Q-Value del estado del entorno en el que se encuentra el agente, suele ser un valor cercano a 0 para que se elija aleatoriamente una acción a llevar a cabo lo menos posible.

**ACTUALIZACIÓN DE LOS Q-VALUES:** Los Q-Values se tienen en cuenta para la toma de decisiones, existe uno por cada acción que el agente puede llevar a cabo en cada estado del entorno, y las acciones que realiza el agente se van seleccionando durante el entrenamiento implementando el Método $\varepsilon$-Greedy. El valor de un Q-Value ligado a una acción "a" en el estado s se actualiza con base en la recompensa adquirida después que el agente realiza la acción "a" en el estado s,

aplicando las Ecuaciones de Bellman y con base en la Recompensa Acumulada con Descuento (ver Figura 6). [18]

**Figura 6.** Ecuaciones de Bellman.

$$Q(s,a) \Leftarrow (1-\alpha)Q(s,a) + \alpha[R + \gamma max_{a'}Q(s',a')] \quad (2)$$

**Fuente:** Reinforcement Learning with TensorFlow A beginner's Guide to designing self-learning systems with TensorFlow and OpenAI Gym (Dutta, 2018).

Los parámetros que las Ecuaciones de Bellman utilizan para su respectiva aplicación en el ámbito del Aprendizaje por Refuerzo son los siguientes:

- $\alpha$:Parámetro learning rate del modelo, hace referencia a la velocidad a la que el agente va aprendiendo (va disminuyendo la pérdida) a medida que se van actualizando los Q-Values y los pesos del modelo. Adquiere un valor entre 0 y 1, de tal forma que entre más pequeño sea el $\alpha$ más lento va aprendiendo el agente (menor es la velocidad de cambio de los pesos del modelo). Es un parámetro que hay que manejar con sumo cuidado debido a que $\alpha$ con valores muy cercanos a 1 también presenta inconvenientes, suele provocar que la función de pérdida no converja al menor valor posible, y eso trae consigo que el modelo no se entrene adecuadamente. [4]

- $Q(s,a)$:Q-Value de determinada acción "a" llevada a cabo en el estado s.

- $R$:Recompensa total y con descuento que se obtiene después de llevar a cabo una acción "a" en el estado s (Rt en la Figura 4).

- $\gamma$: Factor de descuento, en la fórmula multiplica al mayor Q-Value del nuevo estado s' en el que el agente se encontrará después de haber llevado a cabo la acción "a" en el estado s. [18]

**POLÍTICA DE COMPORTAMIENTO:** El objetivo de entrenar un agente inteligente mediante Aprendizaje por Refuerzo es que éste logre aprender a actuar de la mejor manera posible en el momento indicado en busca de maximizar la Recompensa Acumulada con Descuento que se obtiene cada vez que el agente lleva a cabo acciones para desenvolverse en el entorno.

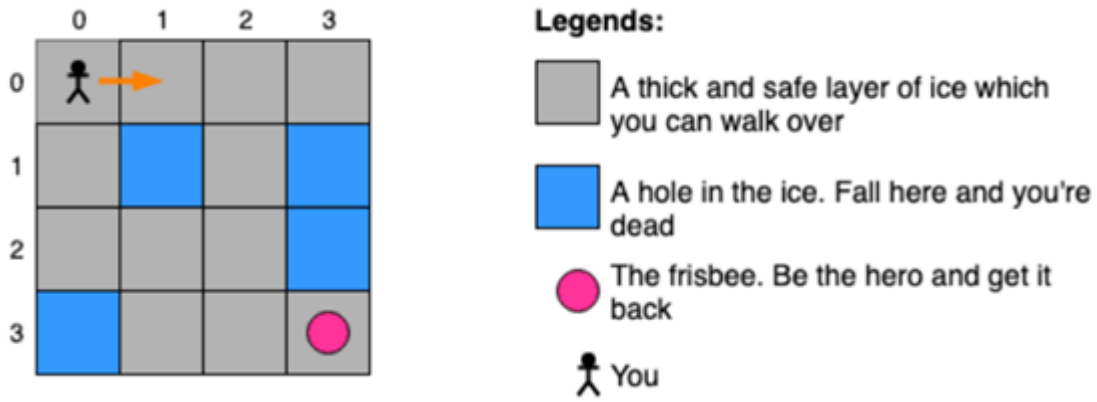
Un agente entrenado adecuadamente es aquel que con base en los Q-Values logra apoyarse en una política de comportamiento óptima, de tal forma que la mayoría de las veces el agente logra evitar las penalizaciones y por el contrario aspira a conseguir la mayor cantidad de bonificaciones posible. La política de comportamiento del agente se va generando a medida que se va entrenando el modelo, de tal manera que el agente va aprendiendo de sus errores y de sus aciertos y se van actualizando los Q-Values del sistema hasta que se considera que se ha optimizado de manera adecuada el comportamiento del agente.[19]

**PROBLEMA DEL LAGO CONGELADO:** Un ejemplo sencillo para comprender de manera adecuada la forma en la que el Aprendizaje por Refuerzo se puede implementar junto al Q-Learning para la resolución de problemas resulta ser el Problema del Lago Congelado (ver Figura 7)

---

[19] NACHUM, Ofir, *et al.* "Bridging the Gap Between Value and Policy Based Reinforcement Learning". eng. En: *arXiv.org* (2017).

**Figura 7.** Problema del Lago Congelado.




**Fuente:** Introduction to Reinforcement Learning: the Frozen Lake Example (Consultado el 08 de Marzo del 2022) Disponible en: https://reinforcement-learning4.fun/2019/06/09/introduction-reinforcement-learning-frozen-lake-example/

El entorno que representa el lago congelado está compuesto por 16 casillas (estados), el agente puede moverse hacia arriba, hacia abajo, hacia la derecha y hacia la izquierda y el objetivo de éste es alcanzar un frisbee que se encuentra en la otra esquina del lago. Hay que tener en cuenta que hay casillas en donde el hielo ya se derritió, y por ende el agente cae al lago al pararse en éstas, recibiendo una penalización de -1 por ello. Por el contrario, obtiene una recompensa de +1 al alcanzar el frisbee.[20]

En un primer momento el agente se encuentra en la esquina superior izquierda del lago congelado, los Q-Values ligados a cada acción que el agente puede realizar para cada estado del entorno empiezan adquiriendo valores aleatorios, y a medida que se va recurriendo al método $\varepsilon$-Greedy para seleccionar acciones a realizar se van actualizando los Q-Values acorde a las recompensas y penalizaciones adquiri-

[20] CHEN, Jim X. "The Evolution of Computing: AlphaGo". eng. En: *Computing in science engineering* 18.4 (2016), págs. 4-7.

das.

De tal manera que si el agente ya ha caído con anterioridad dentro del lago por pisar una de las casillas con hielo derretido, con base en dicha penalización de -1 los Q-Values se actualizan de tal forma que en el futuro es muy poco probable que el agente vuelva a llevar a cabo en el estado adyacente a la casilla con hielo derretido una acción que provoque que vuelva a recibir dicha penalización (debido a que una penalización provoca que el Q-Value ligado a la acción llevada a cabo disminuya su valor considerablemente y la acción más probable a realizar es aquella ligada al mayor Q-Value). Por el contrario, si el agente alcanza el fresbee en la otra esquina del lago y adquiere una recompensa de +1, eso provocará que en partidas futuras sea más probable también alcanzar el frisbee, porque los mayores Q-Values estarán ligados a las acciones que provocaron que el agente adquiriera dicha recompensa.

La política de comportamiento se habrá definido adecuadamente cuando la mayoría de las veces el agente logre alcanzar el frisbee en el entorno del Problema del Lago Congelado, los Q-Values de cada estado provocarán que las acciones llevadas a cabo sigan una política que busca evitar las casillas en las que hay hielo derretido, y por el contrario intenta que el agente alcance el frisbee lo más que pueda.

A continuación se simulará el proceso de entrenamiento llevado a cabo por un Agente Inteligente que busca solucionar el Problema del Lago Congelado entrenándose con base en el Aprendizaje por Refuerzo y el Q-Learning. Se trabajará con $\alpha$=0,1, $\gamma$=0,95 y $\varepsilon$=0.0001, para de esa manera actualizar los Q-Values aplicando las Ecuaciones de Bellman y tomar decisiones con base en el Método $\varepsilon$-Greedy (éste se implementa mediante Colab). Se inicializarán los Q-Values en 0, por lo que durante el primer episodio de entrenamiento las acciones serán llevadas a cabo de manera

aleatoria (cuando por el Método $\varepsilon$-Greedy se opta por realizar aquella acción ligada al mayor Q-Value y existe más de un Q-Value con el mismo máximo valor, se opta por llevar a cabo una acción aleatoria ligada a alguno de los mayores Q-Values que existen para el estado del entorno en el que se encuentra el agente al momento de tomar una decisión).

**Cuadro 1.** Registros Episodio 1

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (0, 1) | No |
| (0, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (0, 2) | No |
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 2) | No |
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (2, 2) | No |
| (0, 0) | Izquierda – Mayor Q-Value | 0 | 0 | 0 | (2, 1) | No |
| (0, 0) | Izquierda – Mayor Q-Value | 0 | 0 | 0 | (2, 0) | No |
| (0, 0) | Abajo – Mayor Q-Value | 0 | -1 | -0.0735 | (3, 0) | No |

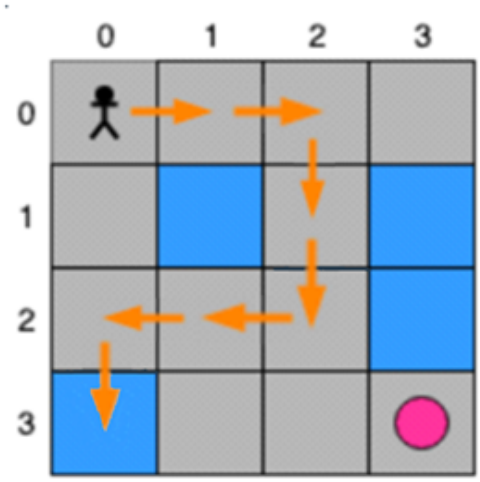
Después de que finaliza el Episodio 1 del entrenamiento, los Q-Values de las ac-

ciones que realizó el agente en cada estado del entorno hasta que éste se reinició (porque el agente pisó una casilla de hielo derretido) se habrán actualizado, de tal forma que en el Episodio 2 el agente tendrá en cuenta los nuevos Q-Values para la toma de decisiones, y el ciclo se repite constantemente durante el entrenamiento.
Para poder visualizar la relación existente entre las acciones que el agente realizó y los estados del entorno en los que las llevó a cabo se recurre a la Q-Table, ésta se va actualizando con los Q-Values para cada dupla Estado-Acción hasta que el entorno se reinicia al finalizar un episodio de entrenamiento.

**Cuadro 2.** Q-Table después del Episodio 1

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | 0 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | 0 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | 0 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | 0 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | 0 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 8.** Ruta Episodio 1.




El entrenamiento se sigue llevando a cabo hasta que el agente alcance el frisbee:

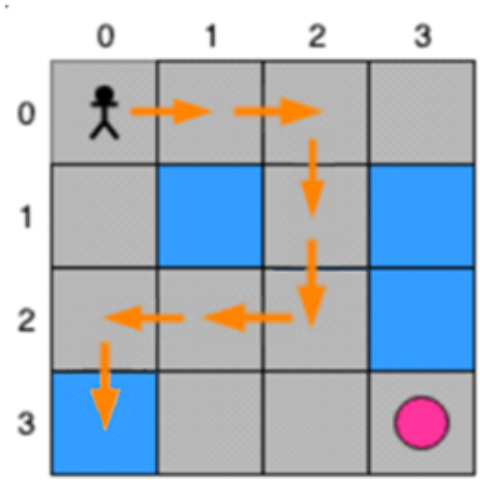
**Cuadro 3.** Registros Episodio 2

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 0) | No |
| (1, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (2, 0) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (2, 1) | No |
| (2, 1) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (2, 2) | No |
| (2, 2) | Derecha – Mayor Q-Value | 0 | -1 | -0.0815 | (2, 3) | Sí |

**Cuadro 4.** Q-Table después del Episodio 2

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | 0 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | 0 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | 0 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | -0.0815 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | 0 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 9.** Ruta Episodio 2.



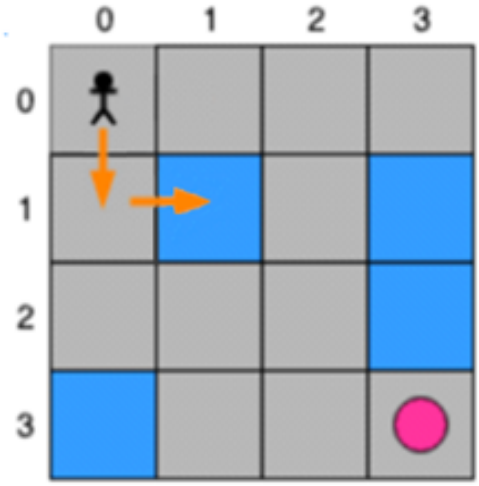
**Cuadro 5.** Registros Episodio 3

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 0) | No |
| (1, 0) | Derecha – Mayor Q-Value | 0 | -1 | -0.095 | (1, 1) | Sí |

**Cuadro 6.** Q-Table después del Episodio 3

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | 0 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | -0.095 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | 0 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | -0.0815 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | 0 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 10.** Ruta Episodio 3.



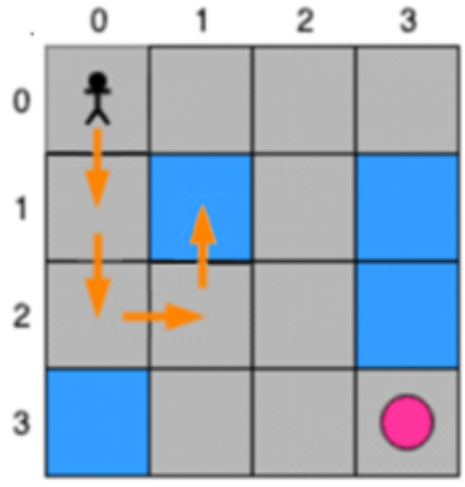
**Cuadro 7.** Registros Episodio 4

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 0) | No |
| (1, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (2, 0) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (2, 1) | No |
| (2, 1) | Arriba – Mayor Q-Value | 0 | -1 | -0.0857 | (1, 1) | Si |

**Cuadro 8.** Q-Table después del Episodio 4

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | 0 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | -0.095 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | -0.0857 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | -0.0815 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | 0 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 11.** Ruta Episodio 4.



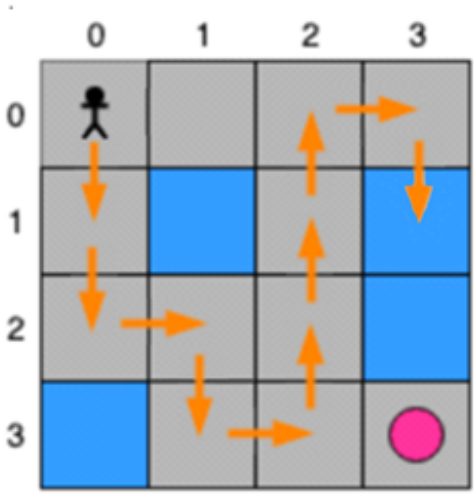
**Cuadro 9.** Registros Episodio 5

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 0) | No |
| (1, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (2, 0) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (2, 1) | No |
| (2, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (3, 1) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (3, 2) | No |
| (2, 0) | Arriba – Mayor Q-Value | 0 | 0 | 0 | (2, 2) | No |
| (2, 0) | Arriba – Mayor Q-Value | 0 | 0 | 0 | (1, 2) | No |
| (2, 0) | Arriba – Mayor Q-Value | 0 | 0 | 0 | (0, 2) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (0, 3) | No |
| (0, 3) | Abajo – Mayor Q-Value | 0 | -1 | -0.063 | (1, 3) | Si |

**Cuadro 10.** Q-Table después del Episodio 5

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | -0.063 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | -0.095 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | -0.0857 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | -0.0815 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | 0 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 12.** Ruta Episodio 5.



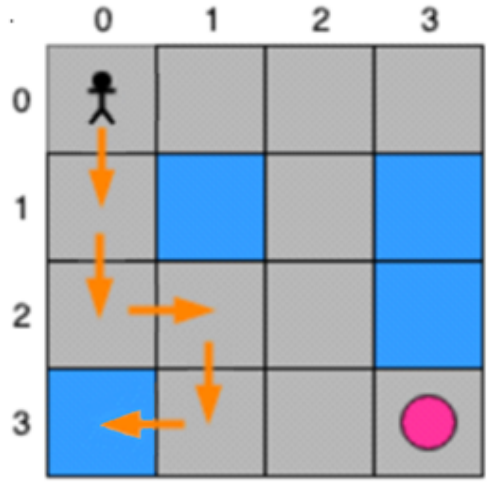
**Cuadro 11.** Registros Episodio 6

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 0) | No |
| (1, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (2, 0) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (2, 1) | No |
| (2, 1) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (3, 1) | No |
| (3, 1) | Izquierda – Mayor Q-Value | 0 | -1 | -0.0815 | (3, 0) | Si |

**Cuadro 12.** Q-Table después del Episodio 6

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | -0.063 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | -0.095 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | -0.0857 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | -0.0815 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | -0.0815 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 13.** Ruta Episodio 6.



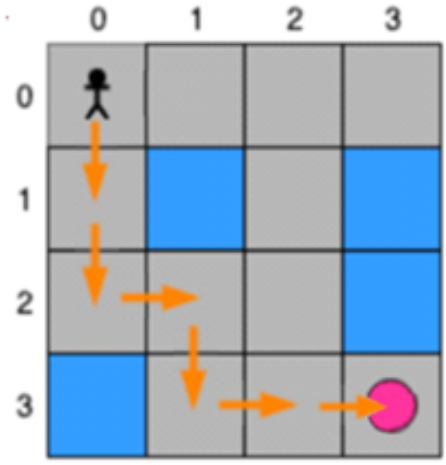
**Cuadro 13.** Registros Episodio 7

| Estado Actual | Acción Realizada | Q-Value Estado-Acción Actual | Recompensa Adquirida (Sin Descuento) | Nuevo Q-Value Estado-Acción Actual | Siguiente Estado | Se Reinicia el Entorno |
|---|---|---|---|---|---|---|
| (0, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (1, 0) | No |
| (1, 0) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (2, 0) | No |
| (2, 0) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (2, 1) | No |
| (2, 1) | Abajo – Mayor Q-Value | 0 | 0 | 0 | (3, 1) | No |
| (3, 1) | Derecha – Mayor Q-Value | 0 | 0 | 0 | (3, 2) | No |
| (3, 2) | Derecha – Mayor Q-Value | 0 | +1 | 0.0774 | (3, 3) | Si |

**Cuadro 14.** Q-Table después del Episodio 7

| Q-Tables | | Acciones | | | |
|---|---|---|---|---|---|
| | | Arriba | Abajo | Izquierda | Derecha |
| Estados | (0, 0) | 0 | 0 | 0 | 0 |
| | (0, 1) | 0 | 0 | 0 | 0 |
| | (0, 2) | 0 | 0 | 0 | 0 |
| | (0, 3) | 0 | -0.063 | 0 | 0 |
| | (1, 0) | 0 | 0 | 0 | -0.095 |
| | (1, 1) | 0 | 0 | 0 | 0 |
| | (1, 2) | 0 | 0 | 0 | 0 |
| | (1, 3) | 0 | 0 | 0 | 0 |
| | (2, 0) | 0 | -0.0735 | 0 | 0 |
| | (2, 1) | -0.0857 | 0 | 0 | 0 |
| | (2, 2) | 0 | 0 | 0 | -0.0815 |
| | (2, 3) | 0 | 0 | 0 | 0 |
| | (3, 0) | 0 | 0 | 0 | 0 |
| | (3, 1) | 0 | 0 | -0.0815 | 0 |
| | (3, 2) | 0 | 0 | 0 | 0.0774 |
| | (3, 3) | 0 | 0 | 0 | 0 |

**Figura 14.** Ruta Episodio 7.

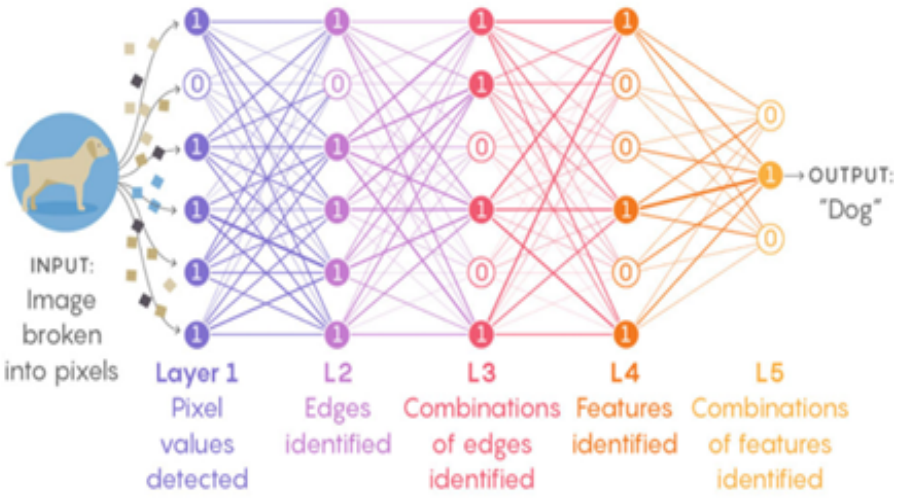



Tras finalizar la simulación se puede llegar a la conclusión de que después de tan solo 7 episodios de entrenamiento el agente fue capaz de alcanzar el frisbee, de-

mostrando de esa manera que el Q-Learning al ser implementado puede utilizarse para generar una adecuada política de comportamiento acorde a la problemática que se busca resolver.

**2.1.5. DEEP LEARNING** Deep Learning es una variante del Machine Learning que se apoya en redes neuronales profundas (con más de una hidden layer) para entrenar modelos, de tal manera que éstos predicen outputs (salidas) dependiendo de los inputs (entradas) que se hayan utilizado para el proceso de entrenamiento (ver Figura 15). Las redes neuronales analizan las entradas, y tal como ocurre en el cerebro humano es necesario que se transmita la información entre neuronas (sinapsis entre layers o capas en términos de Deep Learning) para determinar aquellas características que producen que una determinada entrada genere como salida del modelo un valor en específico (para problemas de clasificación, regresión, para aproximar el valor de los Q-values en el Aprendizaje por Refuerzo, etc). [10]

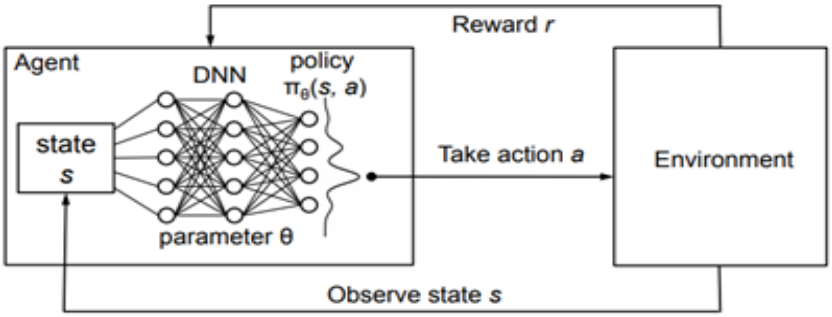
**Figura 15.** Deep Learning.




**Fuente:** New Theory Cracks Open the Black Box of Deep Learning (Consultado el 28 de Mayo del 2021) Disponible en: https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921/

**2.1.6. DEEP Q-NETWORKS** En el caso de las Deep Q-Networks se emplea Q-Learning y Deep Learning (redes neuronales profundas) para diseñar modelos capaces de recibir como entrada el estado actual del entorno para obtener a manera de outputs los Q-Values correspondientes que se tienen en cuenta para la toma de decisiones. Se suele trabajar con CNN's en el ámbito del Aprendizaje por Refuerzo para de esa manera procesar los estados del entorno en formato imagen.[21] Después que el agente selecciona la acción a llevar a cabo, el Q-Value ligado a esta se actualiza, el estado del entorno cambia y el nuevo estado es analizado por el modelo para de esa manera ir actualizando los Q-Values (acorde a las acciones que se van llevando a cabo) de manera iterativa hasta el final del episodio.[22]

**Figura 16.** Deep Q-Network.




**Fuente:** Resource Management with Deep Reinforcement Learning. (Mao et al., 2016

[21] BAKER, Bowen, *et al.* "Designing Neural Network Architectures using Reinforcement Learning". eng. En: *arXiv.org* (2017).

[22] SU, Yishan, *et al.* "DQELR: An Adaptive Deep Q-Network-Based Energy- and Latency-Aware Routing Protocol Design for Underwater Acoustic Sensor Networks". eng. En: *IEEE access* 7 (2019), págs. 9091-9104.

El agente (Agent) va entrenándose episodio tras episodio con base en el modelo y los Q-Values van actualizándose a medida que los episodios van ejecutándose, de tal forma que el agente aprende a comportarse de la mejor manera posible dependiendo del estado (state s) del entorno (Environment) que analice en un momento dado (va guiándose con base en la política de comportamiento a seguir).

La política (policy) $\pi_\theta$(s, a) de comportamiento hace referencia a la capa de salida de la Deep Q-Network (ver Figura 16), representando los diversos Q-Values que existen para cada acción (action a) que el agente puede llevar a cabo si se encuentra en el estado del entorno que es analizado por el modelo. El agente selecciona la acción a realizar apoyándose en el Método $\varepsilon$-Greedy, y el Q-Value de la acción llevada a cabo se actualiza aplicando Ecuaciones de Bellman. De tal manera que los pesos de la red van cambiando (en la Figura 16 los pesos de la red y los hiperparámetros de la misma se ven representados por el atributo parameter $\theta$), generando nuevos Q-values para cada estado del entorno acorde a las recompensas y penalizaciones que se vayan obteniendo. Se puede considerar que el entrenamiento fue llevado de manera exitosa si se logra generar una política de comportamiento adecuada para que el agente actúe de la mejor manera posible, en otras palabras, lo que se busca es obtener los mejores Q-Values para cada estado del entorno, provocando de esa manera que el agente opte la mayoría de las veces por llevar a cabo aquellas acciones que le generan una mayor recompensa acumulada con descuento a largo plazo.[23]

---

[23] MAO, Hongzi, *et al.* "Resource Management with Deep Reinforcement Learning". En: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA: Association for Computing Machinery, 2016, 50–56. DOI: 10.1145/3005745.3005750.

## 2.2. ESTADO DEL ARTE

**2.2.1. DEEPMIND Y ALPHAGO** DeepMind es una empresa británica encargada de trabajar con Inteligencia Artificial capaz de solucionar una gran variedad de problemáticas y de desempeñarse en juegos de diversa índole como lo haría un profesional e incluso mejor.[24] Fue la compañía encargada de diseñar AlphaGo, un agente inteligente capaz de desenvolverse en el juego de mesa Chino Go (un juego mucho más complejo que el ajedrez, puede visualizarse en la Figura 17) de una manera sobrehumana, de tal forma que incluso en el año 2016 logró ganarle al campeón Coreano de Go Lee Sedol, en una competencia de 5 juegos, de los cuales ganó 4.[25]

---

[24] HOLCOMB, Sean D., *et al.* "Overview on DeepMind and Its AlphaGo Zero AI". En: *Proceedings of the 2018 International Conference on Big Data and Education*. ICBDE '18. Honolulu, HI, USA: Association for Computing Machinery, 2018, 67–71. DOI: 10.1145/3206157.3206174.

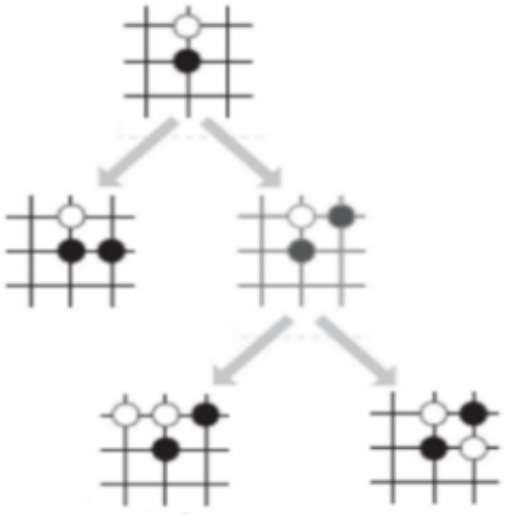
[25] MANEA, Florin; MILLER, Russell G. y NOWOTKA, Dirk, eds. *Sailing Routes in the World of Computation - 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30 - August 3, 2018, Proceedings*. Vol. 10936. Lecture Notes in Computer Science. Springer, 2018. DOI: 10.1007/978-3-319-94418-0.

**Figura 17.** Juego de Mesa Go.

**Fuente:** Google AI in landmark victory over Go grandmaster (Consultado el 16 de Mayo del 2021) Disponible en: https://www.theguardian.com/technology/2016/jan/27/google-hits-ai-milestone-as-computer-beats-go-grandmaster

AlphaGo utilizó un árbol de búsqueda Montecarlo (ver Figura 18) junto a 2 redes convolucionales auxiliares para llevar a cabo el proceso de entrenamiento, se entrenó con base en una gran cantidad de partidas de Go de jugadores profesionales. De tal manera que el agente aprendió qué jugadas son mejores para llevarlas a cabo dependiendo de la situación para que exista una mayor posibilidad de ganarle al contrincante. [24]

**Figura 18.** Árbol de Búsqueda Montecarlo.



**Fuente:** Mastering the game of Go with deep neural networks and tree search (Consultado el 10 de Septiembre del 2021) Disponible en: http://dx.doi.org/10.1038/nature16961

Las redes convolucionales fueron utilizadas para analizar los diversos estados que adquiría un juego de Go e ir actualizando los Q-Values de cada uno de estos a medida que se iban llevando a cabo movimientos (por lo que el entrenamiento se llevó a cabo mediante el Aprendizaje por Refuerzo). El Árbol de Búsqueda Montecarlo (MCTS) fue el algoritmo utilizado para que se realizara el entrenamiento del agente inteligente.[26]

La raíz del árbol hace referencia al estado inicial del juego, posteriormente se seleccionan los nodos hijos (que serían los estados que va adquiriendo el juego a medida que se llevan a cabo movimientos) dependiendo de la cantidad de victorias que se hayan obtenido en el pasado (el nodo hijo más probable a seleccionar es aquel que

[26] HUI, Jonathan. 2018.

a comparación de los otros ha logrado que el agente alcance la mayor cantidad de victorias en el pasado).

Cuando se alcanza una hoja del árbol, si ésta no provoca que se obtenga una victoria o una derrota el árbol de búsqueda se expande, de tal manera que se crean nuevos nodos hijos. Posteriormente, se selecciona alguno y se llevan a cabo acciones aleatorias a partir de dicho nodo hasta que se obtiene un resultado final (una victoria o una derrota).

Con base en dicho resultado se actualiza la información de los nodos anteriores por los que el agente ha pasado, desde el nodo hijo recién creado y seleccionado, hasta la raíz. Posteriormente el agente vuelve a desenvolverse en el árbol de búsqueda con la nueva información adquirida.

El proceso de entrenamiento se llevó a cabo en reiteradas ocasiones hasta que se consideró que se había hallado la política óptima a seguir por parte de un agente inteligente capaz de jugar Go ganando la mayoría de las veces, de esa manera surgió AlphaGo.

AlphaGo resultó ser uno de los primeros agentes inteligentes entrenados con base en Aprendizaje por Refuerzo que acaparó la atención del público y de los medios de comunicación debido a que sirve como prueba evidente de que el futuro de la Inteligencia Artificial va más allá del entendimiento humano y nos supera en casi todos los aspectos. También sirvió para dar a conocer de una manera general el alcance del Aprendizaje por Refuerzo.[27]

---

[27] CHEN, Samuel Yen-Chi, *et al.* "Variational Quantum Circuits for Deep Reinforcement Learning". En: *IEEE Access* 8 (2020), págs. 141007-141024.

**2.2.2. JUEGOS DE ATARI 2600** Agentes inteligentes se pueden entrenar con base en videojuegos clásicos de la consola Atari 2600 por medio del Aprendizaje por Refuerzo, debido a que los sistemas y controles de la época eran sencillos y por ende es posible simularlos de manera no tan compleja en la actualidad (que los entornos y los controles sean sencillos resulta ventajoso para que al trabajar con Aprendizaje por Refuerzo la complejidad computacional al tratar con los datos y llevar a cabo el entrenamiento sea la menor posible).

Los entornos de una gran cantidad de videojuegos del Atari 2600 pueden obtenerse gratuitamente con ayuda de OpenAI Gym, para que con base en éstos, los agentes entrenados mediante Deep Q-Networks aprendan a desenvolverse de la mejor manera posible en dichos videojuegos, siendo los mismos capaces incluso de superar a profesionales enfocados en el área;[28]demostrando de esa manera que el rendimiento de un agente inteligente muchas veces puede superar a los seres humanos incluso en tareas relativamente sencillas como es el caso de jugar un videojuego del Atari 2600 obteniendo el mejor puntaje posible.[29]

---

[28] MNIH, Volodymyr , *et al.* "Playing Atari with Deep Reinforcement Learning". eng. En: *arXiv.org* (2013).

[29] BADIA, Adrià Puigdomènech, *et al.* "Agent57: Outperforming the Atari Human Benchmark". eng. En: *arXiv.org* (2020).

**2.2.3. APRENDIZAJE POR REFUERZO EN IMAGENES** Para una de las implementaciones realizadas durante el desarrollo del proyecto se decidió trabajar con una Deep Recurrent Attention Model[30], un modelo que forma parte del estado del arte en el ámbito del procesamiento de imágenes; combina Recurrent Neural Networks con Aprendizaje por Refuerzo, a la vez que se apoya en una política de Visual Attention para centrarse en las características más relevantes de las imágenes histológicas, para resolver el problema de clasificación de la manera más sencilla posible (identificar si los tejidos presentes en las imágenes poseen o no células cancerosas).

La Deep Recurrent Attention Model está compuesta por 4 Networks relacionadas entre si, la Glimpse Network, la Core Network, la Location Network y la Classification Network.

La Glimpse Network para cada *timestep* t recibe un glimpse $x_t$ en formato imagen (los glimpses se extraen de los parches de las imagenes histológicas que recibe el modelo) y la location $l_t$ de dicho glimpse, que para el primer step de entrenamiento se inicializa en (0,0), representando dicha coordenada el centro de la imagen.

El glimpse $x_t$ es procesado por capas convolucionales, se le lleva a cabo Max Pooling y por último mediante Fully Connected Layers se obtiene el vector $g_{x_t}$. Este vector representa el *qué* de lo procesado por la Glimpse Network.

La tupla location $l_t$ por el contrario fue procesada únicamente por Fully Connected Layers, y de esa manera se obtuvo el vector $g_{l_t}$. Este vector representa el *donde* de lo procesado por la Glimpse Network. Ahora, para combinar el *donde* y el *qué* de lo procesado, es necesario llevar a cabo una multiplicación Element-wise entre los vectores $g_{l_t}$ y $g_{x_t}$, y de esa manera se obtiene el vector $g_t$.

La Core Network es una Recurrent Neural Network compuesta por dos capas, cada

---

[30] MOMENI, Alexandre; THIBAULT, Marc y GEVAERT, Olivier. “Deep Recurrent Attention Models for Histopathological Image Analysis”. En: *bioRxiv* (2018). Preprint. DOI: 10.1101/438341.

una viendose representada por unidades LSTM que codifican los features vectors $r_t^{(1)}$ y $r_t^{(2)}$. El vector $r_t^{(1)}$ recibe como parámetros de entrada el vector $g_t$ que se obtuvo de la Glimpse Network y $r_{t-1}^{(1)}$ para *timesteps* t iguales o mayores a 2 (debido a que se está trabajando con una RNN). Por su parte, el vector $r_t^{(2)}$ recibe $r_t^{(1)}$ y $r_{t-1}^{(2)}$. Se decidió trabajar con unidades LSTM por su capacidad de aprender long-range dependencies y dinámicas de aprendizaje estables.

La Location Network utiliza Fully Connected Layers para procesar el feature-vector $r_t^{(2)}$ y de esa manera predecir la siguiente glimpse location $l_{t+1}$ que debería analizarse del parche que el modelo recibió como entrada (después de que se predice la nueva location inicia un nuevo timestep t de entrenamiento). Debido a que la Location Network es non-differentiable, por lo que no se pueden aplicar técnicas estándar como el back-propagation, se decidió recurrir al Aprendizaje por Refuerzo para entrenar esta red. La finalidad del desarrollo del proyecto como tal es conseguir que la red en cuestión se entrene de la mejor manera posible, debido a que el modelo debe aprender a centrar su atención mediante los glimpses en las áreas de mayor relevancia médica mayoritariamente, para poder clasificar las imágenes de manera adecuada.

La Classification Network, por su parte, se encarga de recibir el feature-vector $r_t^{(1)}$ para predecir de esa manera el label de clasificación que el modelo considera que le corresponde al parche que se recibió como entrada (si el parche presenta tejido sano o canceroso). De esa forma se puede comparar el valor predicho por la Classification Network con el valor label real que se encuentra ligado al parche en cuestión, para obtener una bonificación o una penalización dependiendo de si la predicción fue acertada o no. Dicha recompensa positiva o negativa que se puede llegar a obtener se tiene en cuenta para el método de Aprendizaje por Refuerzo que se implementa para entrenar la Location Network.

El Aprendizaje por Refuerzo en imágenes también se suele implementar al trabajar

con Deep Q-Networks, como en la implementación que se centró en el videojuego de Atari 2600 Pong para el desarrollo de este proyecto. Los estados del entorno se veían representados a manera de imagen, estos eran analizados por la Deep Q-Network para que el agente seleccionara una acción a realizar, y de esa manera se obtenía la bonificación o la penalización necesaria para entrenar el modelo utilizando Aprendizaje por Refuerzo.

La base de datos del MNIST (Modified National Institute of Standards and Technology database) es una enorme base de datos de dígitos escritos manualmente que se suele utilizar para preparar de manera inicial diferentes sistemas y modelos de procesamiento de imágenes**mnist**.

Mediante el dataset MNIST también existen una gran cantidad de implementaciones que se llevaron a cabo recurriendo al Aprendizaje por Refuerzo, tal y como es el caso de la expuesta en el Paper Changing Model Behavior at Test-Time Using Reinforcement Learning**mnist2**. Los resultados obtenidos mediante dicha investigación corroboraron que un modelo ya entrenado con base en el dataset MNIST durante el proceso del testing puede llegar a obtener mejores resultados más rápido si se recurre al Aprendizaje por Refuerzo para mejorar el rendimiento del agente.

## 2.3. HERRAMIENTAS

**2.3.1. LIBRERÍAS PYTHON** Python[31] es el lenguaje de programación que se opta por utilizar para implementar Aprendizaje por Refuerzo mediante Deep Q-Networks, debido a lo útil que resulta ser en el ámbito del Machine Learning por las librerías que se pueden implementar en este lenguaje y por lo intuitivo que es. Las librerías que se utilizaron son las siguientes:

---

[31] VAN ROSSUM, Guido y DRAKE JR, Fred L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

- TensorFlow: Plataforma de código abierto utilizada para implementar aprendizaje automático en el ámbito del desarrollo de inteligencias artificiales.[32] Utilizada para diseñar, compilar y entrenar modelos de redes neuronales, de manera intuitiva y fácil de comprender.

- Numpy: Librería fundamental para trabajar con Python, con ella se crea y procesa información en arreglos n-dimensionales.[33]

- Matplotlib.pyplot: Conjunto de herramientas que permiten generar y/o visualizar las gráficas, las imágenes y las funciones con las que el usuario se encuentra trabajando. También se utiliza para que el usuario pueda visualizar animaciones e incluso interactuar con estas.[34]

- Mitdeeplearning: Librería diseñada por el MIT para simplificar ciertas operaciones relacionadas con el Deep Learning con el propósito de que la codificación sea más intuitiva y menos compleja en dicho ámbito de estudio.[35]

- PyTorch: Framework de código abierto de bajo nivel que se utiliza en el ámbito del Machine Learning para diseñar y entrenar modelos de redes neuronales.[36]

---

[32] ABADI, Martín, *et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[33] HARRIS, Charles R., *et al.* "Array programming with NumPy". En: *Nature* 585.7825 (sep. de 2020), págs. 357-362. DOI: 10.1038/s41586-020-2649-2.

[34] HUNTER, J. D. "Matplotlib: A 2D graphics environment". En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.

[35] MITDEEPLEARNING. *MIT 6.S191: Introduction to Deep Learning*. 2017. URL: https://github.com/aamini/introtodeeplearning (visitado 2022).

[36] PASZKE, Adam, *et al.* "PyTorch: An Imperative Style, High-Performance Deep Learning Library". En: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, págs. 8024-8035.

**2.3.2. GOOGLE COLABORATORY** Herramienta otorgada por Google que permite a los usuarios crear entornos interactivos denominados notebooks, en los cuales se puede escribir en formato texto y también definir celdas de ejecución para programar en Python y en muchos otros lenguajes.[37] Colab permite que se utilicen GPUs presentes en la nube para el entrenamiento de los modelos que se diseñen. Se usa Google Colaboratory para crear los modelos que utilizan los agentes para entrenarse con base en el problema del Péndulo Invertido.

**2.3.3. PYCHARM** PyCharm es un entorno de desarrollo integrado que permite programar en Python para el diseño de productos Software o Machine Learning.[38] Es utilizado para crear y entrenar los modelos que se usan para aplicar el Aprendizaje por Refuerzo en aquellos agentes inteligentes que aprenden a jugar el videojuego de Atari 2600 Pong, y también se utilizó para entrenar la Deep Recurrent Attention Model para la tercera implementación, desarrollada en torno a las imágenes histológicas de próstata.

**2.3.4. OPENAI GYM** OpenAI Gym es un kit de herramientas que se puede utilizar al programar en Python a manera de librería para desarrollar y comparar algoritmos de Aprendizaje por Refuerzo. Se utiliza para entrenar agentes capaces de aprender de manera automática a medida que se desenvuelven en aquellos entornos proporcionados por la toolkit, siendo estos de diversa índole, como es el caso de juegos para el Atari 2600, el sistema del péndulo invertido, agentes que se entrenan en sistemas MuJoCo (simulación avanzada de físicas), etc.
Los entornos con los que se trabaja para el adecuado desarrollo del Proyecto son

---

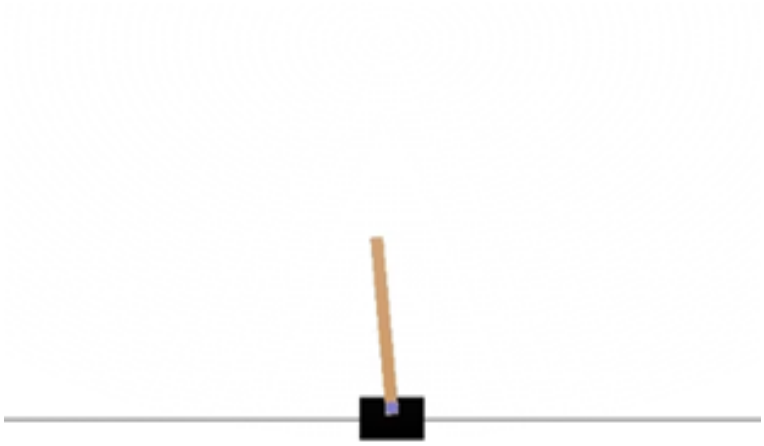
[37] COLABORATORY. *Welcome To Colaboratory*. 2014. URL: `https://colab.research.google.com/notebooks/welcome.ipynb` (visitado 2022).

[38] JETBRAINS. *PyCharm*. 2017. URL: `https://www.jetbrains.com/pycharm/` (visitado 2022).

los siguientes:

- Entorno Péndulo Invertido: El sistema del Péndulo Invertido (ver Figura 19) se conoce en OpenAI Gym como “CartPole-v0”.[39] El agente en dicho entorno adquiere una recompensa de +1 cada timestep que el péndulo se encuentre vertical al eje x de movimiento del agente, el espacio de acción del agente (el total de movimientos que puede llevar a cabo) resulta ser moverse hacia la derecha o hacia la izquierda, y el episodio termina cuando el péndulo se separa 15 grados o más de la posición vertical en la que se busca que se mantenga o cuando el agente se posiciona más de 2,4 unidades del centro del entorno (una unidad en este caso es considerada como el largo del agente). El objetivo del agente es el de mantener el péndulo en equilibrio por 8 segundos, por lo que se puede considerar que el episodio termina también de esa manera.

**Figura 19.** CartPole-v0.

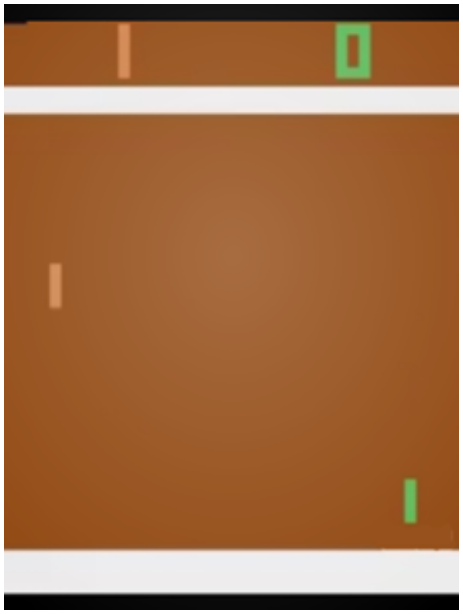


**Fuente:** CARTPOLE-V0. CartPole-v0 [En línea]. [Consultado: 17 de Mayo del 2021]. Disponible en: https://gym.openai.com/envs/CartPole-v0

[39] OPENAIGYM. *CartPole-v0*. 2015. URL: https://gym.openai.com/envs/CartPole-v0 (visitado 2021).

- Entorno Pong: Para el entorno Pong (ver Figura 20) es necesario activar el entorno “Pong-v0” [40] presente en el conjunto de environments que ofrece OpenAI Gym para trabajar con Aprendizaje por Refuerzo. El agente gana una recompensa de +1 cuando logra ganar un punto, por el contrario cuando el contrincante es el que lo adquiere, la recompensa recibida por el agente es de -1 (por ende sería una penalización). El episodio termina cuando alguno de los 2 jugadores (agente o contrincante) logra obtener 21 puntos. El espacio de acción del agente vendría a ser quedarse quieto, moverse hacia la “derecha” (arriba), hacia la “izquierda” (abajo) o llevar a cabo cualquiera de las acciones anteriormente mencionadas a la vez que se está en contacto con el proyectil del juego (por lo que en total el espacio de acción es de 6).

**Figura 20.** Pong-v0.



**Fuente:** PONG-V0. Pong-v0 [En línea]. [Consultado: 17 de Mayo del 2021]. Disponible en: https://gym.openai.com/envs/Pong-v0/

[40] OPENAIGYM. *Pong-V0*. 2015. URL: https://gym.openai.com/envs/Pong-v0/ (visitado 2021).

### 2.3.5. HARDWARE

- Máquina compartida del grupo HDSP[41]: Se utiliza la GPU (una tarjeta gráfica GeForce RTX 3090) de la máquina para entrenar los agentes que aprenden a jugar el videojuego de Atari 2600 Pong, y los scripts en Python son ejecutados mediante PyCharm. Se accede remotamente a la máquina mediante AnyDesk[42].

- Computadores personales: Utilizados para ejecutar los notebooks que se diseñan mediante Colab (Péndulo Invertido) y para obtener los parches con base en las imágenes histológicas mediante los cuales se entrenó la Deep Recurrent Attention Model.

- Equipo compartido del grupo BIVL2AB: Utilizado para entrenar la Deep Recurrent Attention Model mediante los parches que se extrajeron desde el computador personal utilizado para ello.

---

[41] HDSP. *Grupo de investigación en diseño de algoritmos y procesamiento de datos multidimensionales*. 2022. URL: http://cormoran.uis.edu.co/eisi/grupo/hdsp (visitado 2021).

[42] GMBH, AnyDesk Software. *AnyDesk*. 2012. URL: https://anydesk.com/es (visitado 2021).

# 3. METODOLOGÍA

## METODOLOGÍA PARA EL DESARROLLO DEL PROYECTO

Para el desarrollo del Proyecto es necesario dividir el trabajo en 3 fases fundamentales a tener en cuenta para la adecuada realización de éste:

| | |
|---|---|
| Fase 1: Construcción de los Modelos | En primer lugar se diseñan las Redes Neuronales (la mayoría de las mismas siendo Deep Q-Networks) que se utilizarán para entrenar a los Agentes Inteligentes. |
| Fase 2: Entrenamiento de los Modelos | Luego se recurre al Aprendizaje por Refuerzo para que los Agentes aprendan a adquirir las mejores recompensas posibles en los entornos en los que se desenvuelven. |
| Fase 3: Comparar desempeño de los Agentes | Después del entrenamiento de los Agentes se visualizarán las gráficas que representan los puntajes o las recompensas adquiridas por estos a medida que se iban entrenando desenvolviéndose en sus respectivos entornos, y se compararán resultados entre Agentes Inteligentes entrenados con base en Redes Neuronales distintas para cada caso de estudio. |

# 4. PÉNDULO INVERTIDO

En este capítulo se describe a detalle la implementación del código al que se recurrió para dar solución al problema del Péndulo Invertido, utilizando Agentes Inteligentes que se entrenaron con base en el Aprendizaje por Refuerzo. Parte del código se adaptó de un Notebook en Colab diseñado por un grupo de investigación del Massachusetts Institute of Technology.[43]

## 4.1. CONSTRUCCIÓN DE LOS MODELOS

En esta etapa del desarrollo se diseñaron las Redes Neuronales que fueron utilizadas por los Agentes para entrenarse de tal forma que lograron superar el problema del Péndulo Invertido. Las Redes Neuronales que se construyeron fueron diseñadas mediante la API Keras[44] ejecutándose sobre TensorFlow.

**4.1.1. MODELO DE REFERENCIA** El Modelo de referencia que se utilizó en el Notebook del MIT estaba compuesto únicamente por dos capas densas (ver Figura 21). Debido a la sencillez de la problemática en cuestión se optó por utilizar dicho tipo de capa ya que resulta ser el pilar básico del desarrollo de una Red Neuronal.[45] La función de activación ReLU fue utilizada en la capa de entrada de la red que se encargaba de recibir los valores numéricos que hacían referencia a los estados del entorno en los que se desenvolvía el Agente durante el entrenamiento. Se decidió

---

[43] MIT. *6.S191 Introduction to Deep Learning*.

[44] CHOLLET, Francois, *et al. Keras*. 2015. URL: https://github.com/fchollet/keras.

[45] VERMA, Yugesh. *A Complete Understanding of Dense Layers in Neural Networks*. 2021. URL: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/.

utilizar dicha función de activación debido a que resulta ser la más común a la que se recurre al momento de diseñar una Red Neuronal.[46]

**Figura 21.** Modelo de referencia Péndulo Invertido.

```
[ ]  def create_cartpole_model():
       model = tf.keras.models.Sequential([

           tf.keras.layers.Dense(units=32, activation='relu'),


           tf.keras.layers.Dense(units=n_actions, activation=None)

       ])
       return model

     cartpole_model = create_cartpole_model()
```

En la capa de salida carecía de sentido utilizar la función de activación ReLU, debido a que el rango de esta función se encuentra distribuido en el intervalo de [0, $\infty$ ), de tal manera que al recibir un valor negativo como entrada, la salida es cero.[47] Por ende no es utilizada en la capa final de las Q-Networks, ya que en reiteradas ocasiones los Q-Values adquieren valores negativos. Por dicho motivo no se utiliza función de activación alguna en la capa de salida de la red.

La capa de entrada de la red posee 32 neuronas (se suele trabajar para una mayor estabilidad con un número de neuronas en cada capa igual a una potencia de 2), y

---

[46] CHEN, Bindi. *Why Rectified Linear Unit (ReLU) in Deep Learning and the best practice to use it with TensorFlow*. 2021. URL: https://towardsdatascience.com/why-rectified-linear-unit-relu-in-deep-learning-and-the-best-practice-to-use-it-with-tensorflow-e9880933b7ef.

[47] BROWNLEE, Jason. *A Gentle Introduction to the Rectified Linear Unit*. 2019. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

la capa de salida tiene únicamente dos neuronas, haciendo referencia a los valores que llegan a adquirir los Q-Values de las dos únicas acciones (moverse hacia la derecha o hacia la izquierda) que puede llevar a cabo el agente en el entorno del Péndulo Invertido (ver Figura 22).

**Figura 22.** Espacio de acción Péndulo Invertido.

```
[ ]  env = gym.make("CartPole-v0")
     env.seed(1)
     n_actions = env.action_space.n
     print("Número posible de acciones que el agente puede llevar a cabo:", n_actions)

     Número posible de acciones que el agente puede llevar a cabo: 2
```

El modelo de referencia en cuestión fue la única red neuronal con la que se trabajó durante el desarrollo del Proyecto que no era una Deep Q-Network, únicamente se puede considerar una Q-Network, debido a que no se utilizaron hidden layers (capas intermedias) y la estructura de la red sólo estaba compuesta por una capa de entrada y otra de salida.

**4.1.2. MODELO PROPUESTO** El modelo que se propuso (ver Figura 23) estaba compuesto por 4 capas densas, se recurrió a la función de activación ReLU en todas las capas a excepción de la última y las neuronas se distribuyeron de manera descendente.

El diseño de la red se llevó a cabo de dicha forma debido a que en el ámbito del Machine Learning para mejorar el performance de una red neuronal de carácter no profundo se suele recurrir al Deep Learning mediante la implementación de hidden layers.[48]

---

[48] MAHAPATRA, Sambit. *Why Deep Learning over Traditional Machine Learning?* 2018. URL: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063.

**Figura 23.** Modelo propuesto Péndulo Invertido.

```
[ ]  def create_model2():
       model = tf.keras.models.Sequential([

           tf.keras.layers.Dense(units=64, activation='relu'),

           tf.keras.layers.Dense(units=32, activation='relu'),

           tf.keras.layers.Dense(units=16, activation='relu'),

           tf.keras.layers.Dense(units=n_actions, activation=None)

       ])
       return model

     cartpole_model = create_model2()
```

## 4.2. ENTRENAMIENTO DE LOS MODELOS

Haciendo uso del Aprendizaje por Refuerzo los agentes aprendieron a desenvolverse de la mejor forma posible en sus respectivos entornos, dependiendo de los modelos de Q-Networks que se utilizaron para el entrenamiento. El proceso de entrenamiento fue llevado a cabo mediante la implementación de una gran cantidad de algoritmos, que utilizados en conjunto sirvieron para entrenar agentes inteligentes capaces de superar el entorno del Problema del Péndulo Invertido disponible en OpenAI Gym, mediante el uso del Aprendizaje por Refuerzo.

**4.2.1. SELECCIONAR ACCIÓN** Para que el agente seleccionara las acciones que realizó durante el entrenamiento se recurrió a la función choose_action (ver Figura 24). Los parámetros de entrada utilizados fueron el modelo de red neuronal que se usó para el Q-Learning, el observation o el conjunto de observations (estados del entorno) que analizó el agente antes de tomar una decisión (seleccionar qué acción llevar a cabo) y una variable bandera que se utilizó para saber si se iban a seleccionar acciones con base en una única observation o en un conjunto de estas.

**Figura 24.** Función choose_action Péndulo Invertido.

```
[ ]  def choose_action(model, observation, single=True):
       observation = np.expand_dims(observation, axis=0) if single else observation


       logits = model.predict(observation)


       action = tf.random.categorical(logits, num_samples=1)


       action = action.numpy().flatten()

       return action[0] if single else action
```

Si el modelo recibía como entrada un estado del entorno la salida correspondiente era el valor de los Q-Values para el estado en cuestión, y dependiendo de dichos valores era seleccionada una acción al azar, teniendo en cuenta que la acción más probable a realizar era aquella ligada al mayor Q-Value (en este caso en vez de recurrir al Método $\varepsilon$-Greedy se utilizó el tf.random.categorical).

**4.2.2. DISEÑO DE LA MEMORIA DEL AGENTE** La memoria del agente se simuló mediante la clase Memory tal y como se expone en la Figura 25. Fue utilizada para almacenar los estados del entorno que alcanzó el agente, junto a las acciones que llevó a cabo en cada uno de estos y las recompensas que obtuvo como conse-

cuencia de ello. Con base en la información adquirida se entrenaron las Q-Networks para actualizar los Q-Values de los estados del entorno.

**Figura 25.** Memoria del agente Péndulo Invertido.

```
[ ]  class Memory:
       def __init__(self):
           self.clear()


       def clear(self):
           self.observations = []
           self.actions = []
           self.rewards = []


       def add_to_memory(self, new_observation, new_action, new_reward):
           self.observations.append(new_observation)

           self.actions.append(new_action)


           self.rewards.append(new_reward)



     def aggregate_memories(memories):
       batch_memory = Memory()

       for memory in memories:
         for step in zip(memory.observations, memory.actions, memory.rewards):
           batch_memory.add_to_memory(*step)

       return batch_memory


     memory = Memory()
```

**4.2.3. RECOMPENSA CON DESCUENTO** Se recurrió a la función discount_rewards (ver Figura 26) para obtener las recompensas acumuladas y con descuento (que se utilizaron para actualizar los Q-Values) con base en las recompensas que se almacenaron en memoria durante el proceso de entrenamiento. gamma representa el factor de descuento, y las recompensas con descuento se normalizaron para que se mantuviera constante una escala en común,[49] con el propósito de obtener los mejores resultados posibles al momento de entrenar el agente.

**Figura 26.** Recompensa con descuento Péndulo Invertido.

```
[ ]  def normalize(x):
       x -= np.mean(x)
       x /= np.std(x)
       return x.astype(np.float32)


     def discount_rewards(rewards, gamma=0.95):
       discounted_rewards = np.zeros_like(rewards)
       R = 0
       for t in reversed(range(0, len(rewards))):

           R = R * gamma + rewards[t]
           discounted_rewards[t] = R

       return normalize(discounted_rewards)
```

[49] JAITLEY, Urvashi. *Why Data Normalization is necessary for Machine Learning models*. 2018. URL: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

**4.2.4. CÁLCULO DE LA PÉRDIDA** Para calcular la pérdida que se buscaba minimizar (entre menor sea la pérdida, mejor es la eficiencia de la red neuronal) se recurrió a la función compute_loss (ver Figura 27). Recibía como parámetros de entrada las acciones y recompensas que se almacenaron en memoria, junto a los Q-Values que se obtuvieron mediante la Q-Network que recibía como entrada los estados del entorno que también se habían guardado en memoria.

**Figura 27.** Cálculo de la pérdida Péndulo Invertido.

```
[ ]  def compute_loss(logits, actions, rewards):

       neg_logprob = tf.nn.sparse_softmax_cross_entropy_with_logits(
           logits=logits, labels=actions)



       loss = tf.reduce_mean( neg_logprob * rewards )

       return loss
```

Se trabajó con una pérdida en formato Cross Entropy Loss,[50] que se utilizó para comparar los Q-Values predichos por la Q-Network (logits), con las acciones que se realizaron durante el proceso de entrenamiento (cada acción estaba ligada a un valor de 1 ó 0, dependiendo de si se llevó a cabo o no respectivamente en los estados del entorno que la Q-Network recibió como parámetros de entrada para calcular los Q-Values).

La pérdida general que se obtuvo mediante la función compute_loss se calculó mediante la media de los productos entre las pérdidas de la Q-Network que recibió los

[50] DAHAL, PARAS. *Classification and Loss Evaluation - Softmax and Cross Entropy Loss*. 2018. URL: https://deepnotes.io/softmax-crossentropy.

estados del entorno que se almacenaron en memoria y las recompensas (con descuento) que durante el proceso de entrenamiento se obtuvieron después de que el agente analizara dichos estados y realizara acciones.

**4.2.5. FUNCIÓN DE ENTRENAMIENTO** Para entrenar la red neuronal es necesario recurrir a la función train_step (ver Figura 28). Esta recibe como parámetros de entrada el modelo de Q-Network, un optimizer que se utiliza para ir alterando los valores de los pesos de la red, el learning rate y demás atributos de la misma en busca de reducir la pérdida lo más que se pueda;[51] y las observaciones, acciones y recompensas (con descuento) que se almacenaron en memoria.

**Figura 28.** Función de entrenamiento Péndulo Invertido.

```
[ ]  def train_step(model, optimizer, observations, actions, discounted_rewards):
       with tf.GradientTape() as tape:

           logits = model(observations)


           loss = compute_loss(logits, actions, discounted_rewards)



       grads = tape.gradient(loss, model.trainable_variables)

       optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

[51] DOSHI, Sanket. *Various Optimization Algorithms For Training Neural Network*. 2019. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

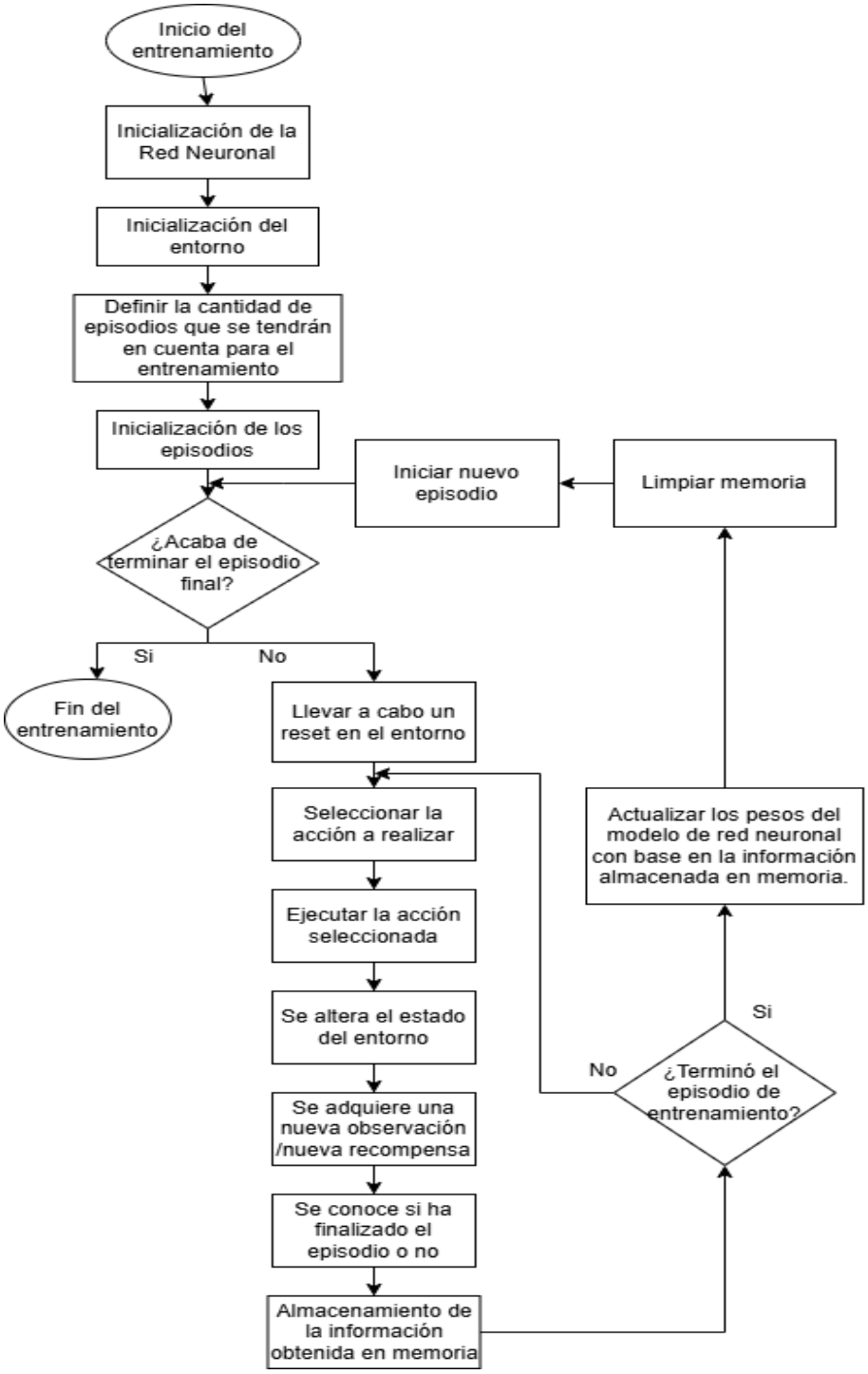
Las observaciones se utilizaron para predecir los Q-Values con ayuda de la Q-Network, estos junto a las acciones y recompensas (con descuento) que se almacenaron en memoria sirvieron para calcular la pérdida que se tiene en cuenta para el proceso de entrenamiento de la red neuronal, mediante la función compute_loss.

La función tf.GradientTape().gradient() recibió como parámetros de entrada la pérdida y las trainable variables de la red (aquellos atributos que mediante el entrenamiento cambiarán su valor, como es el caso de los pesos de la Q-Network). Fue utilizada para aplicar el Método del Gradiente Descendente[52] con base en un optimizer (que se usa en la función optimizer.apply_gradients), con el propósito de alterar los pesos de la red en busca de la menor pérdida posible a medida que esta se iba entrenando.

**4.2.6. ALGORITMO GENERAL** Los modelos de las redes neuronales se entrenaron teniendo en cuenta el diagrama de flujo expuesto en la Figura 29.

[52] EDUCATION, IBM Cloud. *What is Gradient Descent?* 2020. URL: https://www.ibm.com/cloud/learn/gradient-descent.

**Figura 29.** Entrenamiento de los Modelos Péndulo Invertido.



El algoritmo general utilizado era el eje fundamental al que fue necesario recurrir para entrenar agentes inteligentes capaces de superar el Problema del Péndulo Invertido mediante el uso del Aprendizaje por Refuerzo (ver Figura 30). En dicho algoritmo en primer lugar se definió el learning rate, este puede adquirir un valor entre 1 y 0, y representa la velocidad a la que el modelo se adapta al problema.[53]

**Figura 30.** Algoritmo general Péndulo Invertido.

```
[ ]  learning_rate = 1e-3
     optimizer = tf.keras.optimizers.Adam(learning_rate)

     cartpole_model = create_cartpole_model()

     smoothed_reward = mdl.util.LossHistory(smoothing_factor=0.9)
     plotter = mdl.util.PeriodicPlotter(sec=2, xlabel='Iterations', ylabel='Rewards')

     if hasattr(tqdm, '_instances'): tqdm._instances.clear()
     for i_episode in range(500):

       plotter.plot(smoothed_reward.get())

       observation = env.reset()

       while True:

           action = choose_action(cartpole_model, observation)
           next_observation, reward, done, info = env.step(action)

           memory.add_to_memory(observation, action, reward)

           if done:

               total_reward = sum(memory.rewards)
               smoothed_reward.append(total_reward)

               train_step(cartpole_model, optimizer,
                          observations=np.vstack(memory.observations),
                          actions=np.array(memory.actions),
                          discounted_rewards = discount_rewards(memory.rewards))

               memory.clear()
               break

           observation = next_observation
```

[53] BROWNLEE, Jason. *Understand the Impact of Learning Rate on Neural Network Performance*. 2020. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

Mediante la Figura 31 se puede visualizar una comparación entre los diversos comportamientos que presenta el Método del Gradiente Descendente con diferentes valores para el learning rate. Utilizando uno muy pequeño se requiere de una gran cantidad de episodios de entrenamiento para que se alcance el valor mínimo de pérdida que se espera obtener. Con un valor adecuado y óptimo para el learning rate es posible que dicho mínimo se alcance con una menor cantidad de episodios de entrenamiento, y si por el contrario se prueba con un learning rate muy cercano a 1 es muy probable que el cambio brusco que se produce sobre los pesos de la red traiga consigo un comportamiento divergente (no se alcanza el mínimo). [53]

**Figura 31.** Comparación Learning Rate.


**Fuente:** JORDAN, Jeremy. Setting the learning rate of your neural network (Consultado el 08 de Junio del 2022) Disponible en: https://www.jeremyjordan.me/nn-learning-rate/

Se utilizó el optimizer Adam debido a que es útil para la optimización estocástica, es computacionalmente eficiente y los requerimientos necesarios para su implementación no utilizan mucho espacio en memoria.[54] También se recurrió a funciones propias del grupo de investigación del MIT (mdl.util.LossHistory y mdl.util.PeriodicPlotter) para que durante el proceso de entrenamiento se pudiera visualizar una gráfica en la que el eje x hacía referencia a las iteraciones o episodios que se tuvieron en cuenta para entrenar el agente y el eje y representaba las recompensas que se obtuvieron al final de cada episodio. Para visualizar la variación de la recompensa de la mejor forma posible se decidió "suavizar" la gráfica, de tal forma que las correlaciones existentes entre iteraciones y recompensas adquiridas se transformaron en una función continua.

Dependiendo de la cantidad de episodios de entrenamiento que se definieron, se recurrió a un bucle for para que en cada uno de estos en primer lugar se llevara a cabo un reset en el entorno, para posteriormente recurrir a la función choose_action con el propósito de seleccionar la acción que el agente llevaba a cabo. Dependiendo de su elección se adquiría una nueva observación (un nuevo estado del entorno), una recompensa e información adicional, y en memoria se almacenaba la acción que se seleccionó mediante la función choose_action, el estado del entorno en el que esta se realizó y la recompensa que se obtuvo como consecuencia de ello.

Luego, el ciclo se repitió desde la selección de la acción a realizar, pero esta vez el estado del entorno que la función choose_action recibió como parámetro de entrada era la nueva observación que se había adquirido en el timestep anterior, y el proceso anteriormente descrito se llevó a cabo continuamente hasta que terminó el episodio de entrenamiento.

---

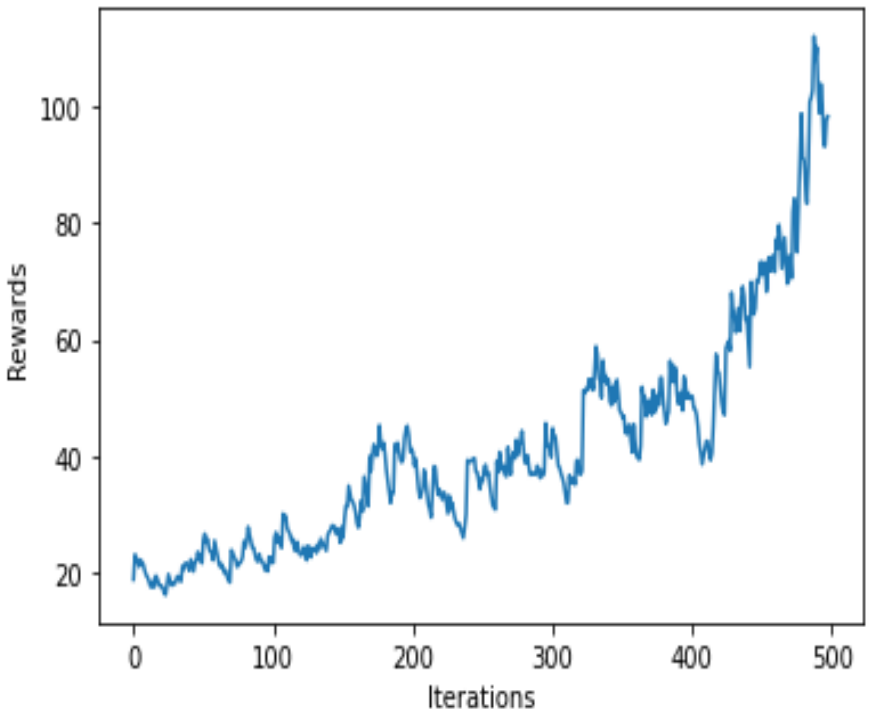
[54] BROWNLEE, Jason. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. 2017. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

Posteriormente, la información almacenada en memoria se utilizó para entrenar la red neuronal mediante la función train_step, y después se limpió la memoria y se llevó a cabo un reset en el entorno, para que el entrenamiento del agente continuara con base en los siguientes episodios.

### 4.3. COMPARAR DESEMPEÑO DE LOS AGENTES

Tras haber llevado a cabo el entrenamiento de los agentes, es necesario comparar resultados mediante las gráficas que se fueron generando a medida que las IA's se iban desenvolviendo en sus respectivos entornos, y utilizando videos en los que se pueden visualizar el desempeño de los agentes.
Los resultados que se obtuvieron con base en el modelo de referencia se encuentran expuestos en la Figura 32.

**Figura 32.** Resultados Péndulo Invertido modelo de referencia.

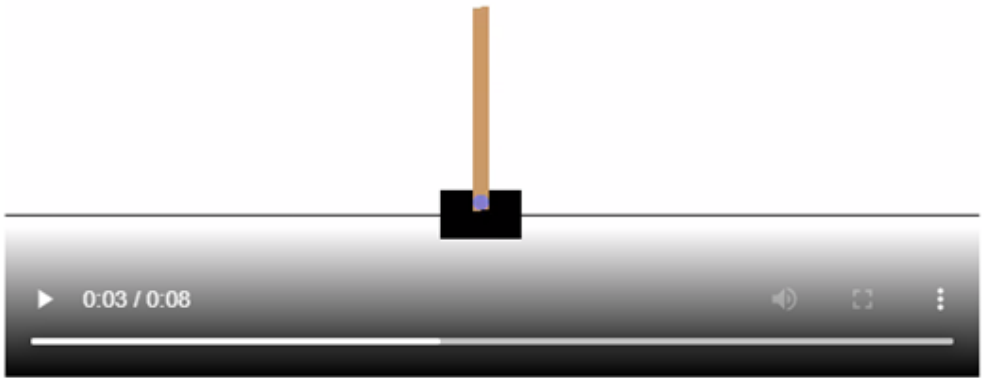


Utilizando 500 episodios de entrenamiento, el modelo de referencia le permitió al agente ir aumentando paulatinamente las recompensas que iba adquiriendo. Analizando la tendencia de la gráfica se puede llegar a la conclusión de que probablemente con más episodios de entrenamiento, el rendimiento del agente hubiera podido ser mucho mejor.

Se recurrió a funciones propias del grupo de investigación del MIT para poder generar el video en el que se puede visualizar el desempeño de un agente inteligente en un determinado entorno de OpenAI Gym, con base en una red neuronal ya entrenada, tal y como se puede visualizar en la Figura 33.

**Figura 33.** Función generar video Péndulo Invertido.

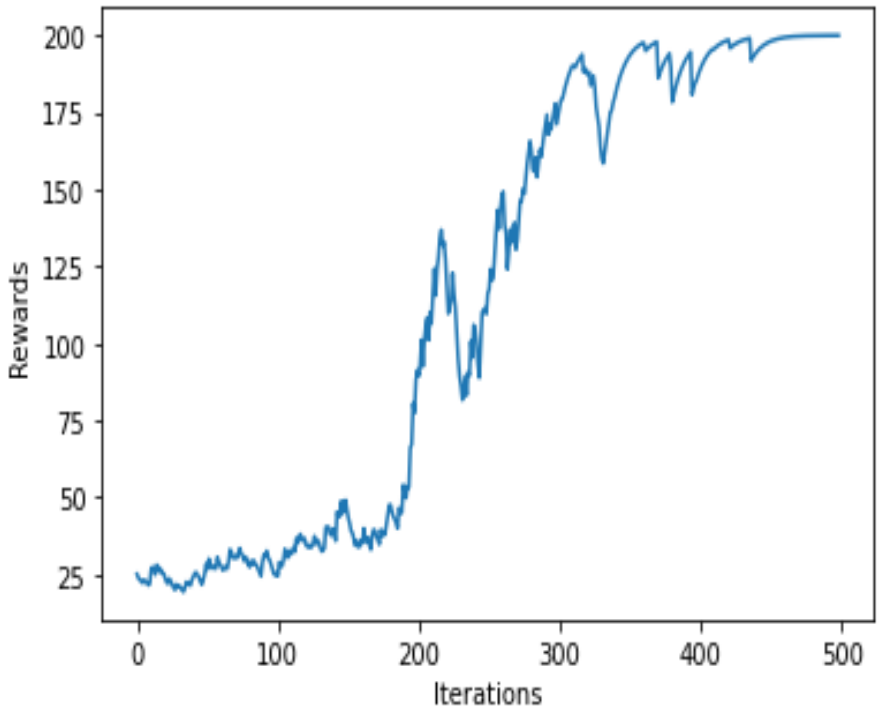
```
[ ]  saved_cartpole = mdl.lab3.save_video_of_model(cartpole_model, "CartPole-v0")
     mdl.lab3.play_video(saved_cartpole)

     Successfully saved 200 frames into CartPole-v0.mp4!
```



En el video[55] generado con base en el modelo de referencia el agente logra superar el Problema del Péndulo Invertido, manteniéndolo estable durante 8 segundos y centrado, el movimiento es casi nulo.

Mediante la Figura 34 también es posible visualizar la gráfica que representa la relación existente entre recompensas adquiridas y episodios de entrenamiento del agente que se desenvolvió en el entorno del Problema del Péndulo Invertido con base en el modelo propuesto.

**Figura 34.** Resultados Péndulo Invertido modelo propuesto.




A diferencia de los resultados obtenidos con el modelo de referencia, en este caso las recompensas adquiridas por el agente sobrepasan en gran medida a aquellas que se alcanzaron apoyándose en dicho modelo. Teniendo en cuenta los mismos

[55] RODRÍGUEZ, Julián. *Péndulo Invertido*. 2022. URL: https://youtu.be/h3AQCP4I0Vg.

500 episodios que en el caso anterior, con base en la red neuronal propuesta se obtuvo aproximadamente el doble del puntaje adquirido por el agente que se entrenó apoyándose en el modelo de referencia. Con la notable diferencia de que en este caso el potencial de la red propuesta impide que el agente alcance un mayor puntaje (la gráfica no tiende a infinito, tiende a 200).

En el video[56] generado con base en el modelo propuesto se logra visualizar al agente moviéndose hacia la derecha, a medida que mantiene estable el péndulo invertido. A diferencia del anterior video, en este caso el agente no logra mantenerse igual de centrado, pero aún así se obtuvo una mayor recompensa debido a que lo único que le otorga recompensas positivas resulta ser mantener estable el péndulo (sin importar que el agente se mueva o no).

[56] RODRÍGUEZ, Julián. *Péndulo Invertido Red Propuesta*. 2022. URL: https://youtu.be/v1qSojOonhs.

# 5. PONG

Entrenando agentes inteligentes es posible que estos aprendan a superar al contrincante en el videojuego de Atari 2600 Pong. Para ello se recurrió al Q-Learning con base en Q-Networks, y se utilizó el entorno “Pong-v0” disponible en OpenAI Gym para el entrenamiento de los modelos. Parte del código fue adaptado del mismo Notebook sobre el cual se trabajó durante el desarrollo del Capítulo 4.

## 5.1. CONSTRUCCIÓN DE LOS MODELOS

**5.1.1. MODELO DE REFERENCIA.** La red neuronal que fue utilizada como referencia por el grupo de investigación del MIT se puede visualizar mediante la Figura 35.

**Figura 35.** Modelo de referencia Pong.

```
[ ]  Conv2D = functools.partial(tf.keras.layers.Conv2D, padding='same', activation='relu')
     Flatten = tf.keras.layers.Flatten
     Dense = tf.keras.layers.Dense


     def create_pong_model():
       model = tf.keras.models.Sequential([

         Conv2D(filters=32, kernel_size=5, strides=2),
         Conv2D(filters=48, kernel_size=5, strides=2),

         Conv2D(filters=64, kernel_size=3, strides=2),
         Conv2D(filters=64, kernel_size=3, strides=2),
         Flatten(),
         Dense(units=128, activation='relu'),

         Dense(units=n_actions, activation=None)


       ])
       return model


     pong_model = create_pong_model()
```

Se utilizó keras para diseñar una red neuronal convolucional,[57] debido a que en el contexto del videojuego Pong es óptimo analizar los estados del entorno en formato imagen, para que el output de la red haga referencia a los Q-Values que se tuvieron en cuenta para la toma de decisiones. Los filtros se definieron de manera ascendente para cada capa, exceptuando la última, debido a que esta en el ámbito de las Q-Networks genera como salida los Q-Values que se encuentran ligados a las acciones que el agente puede llevar a cabo (en este caso, tal y como se puede visualizar en la Figura 36 y en el Capítulo 2.3.4, se pueden realizar únicamente 6 acciones).

**Figura 36.** Espacio de acción Pong.

```
[ ]  def create_pong_env():
       return gym.make("Pong-v0", frameskip=5)
     env = create_pong_env()
     env.seed(1);

     n_actions = env.action_space.n
     print("Number of possible actions that the agent can choose from =", n_actions)

     Number of possible actions that the agent can choose from = 6
```

Las funciones de activación para las capas se asignaron de igual manera que en las implementaciones que se desarrollaron en torno al Problema del Péndulo Invertido, se utilizó la función relu en todas las capas a excepción de la última.

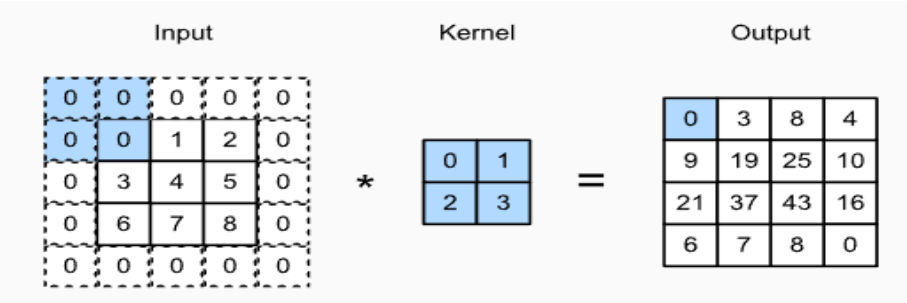
El padding de las convolutional layers fue del tipo same, de tal forma que las matrices numéricas que recibían como entrada (los píxeles de la imagen que recibe la red como input se ven representados también a manera de matriz) se transformaban y

---

[57] SAHA, Sumit. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

aumentaban su tamaño al adquirir un perímetro de valores 0[58] (tal y como se puede visualizar en el input de la Figura 37, en el que se utiliza padding en una matriz 3x3).

**Figura 37.** Capa convolucional padding y kernel.




**Fuente:** Padding and Stride (Consultado el 09 de Julio del 2022) Disponible en: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html

Se recurrió al padding para que no falte píxel alguno por analizar al momento de utilizar el kernel para generar el output de la capa convolucional, debido a que es importante que los márgenes de la imagen que recibe la red como entrada se procesen adecuadamente.[59]

Tal y como se puede ver en la Figura 37 (se detalla mejor en la Figura 38), el kernel es una matriz utilizada para llevar a cabo productos matriciales con base en el input de la capa convolucional, de tal forma que se va generando una matriz numérica a manera de output a medida que el kernel se va utilizando (lo más óptimo es que se alcancen a procesar todos y cada uno de los valores de la matriz que recibe la capa como entrada).[60]

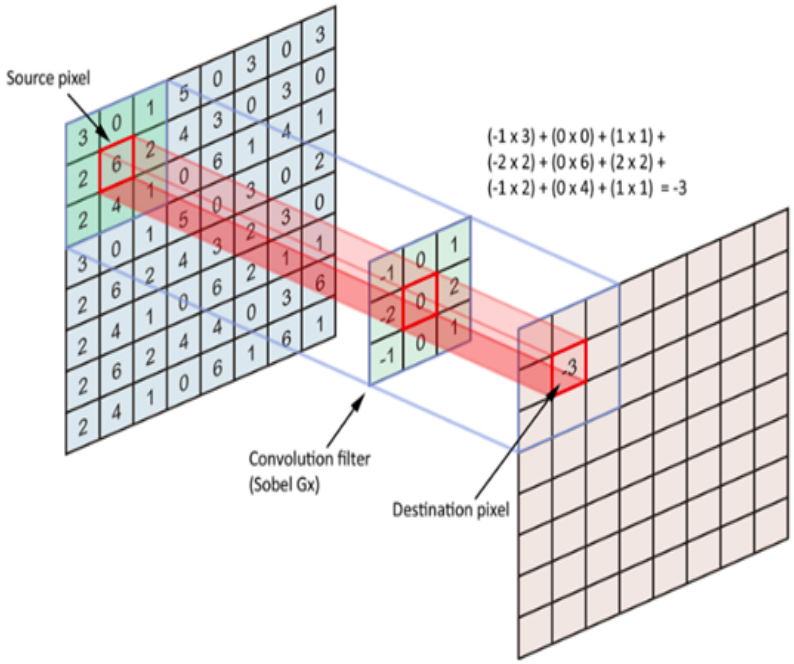
---

[58] *Conv2D layer*. URL: https://keras.io/api/layers/convolution_layers/convolution2d/.

[59] ZHANG, Aston, *et al. DIVE INTO DEEP LEARNING*. 2020. URL: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html.

[60] GANESH, Prakhar. *Types of Convolution Kernels : Simplified*. 2019. URL: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37.

**Figura 38.** Producto Kernel Input Capa convolucional.




**Fuente:** Understanding Deep Self-attention Mechanism in Convolution Neural Networks (Consultado el 16 de Julio del 2022) Disponible en: https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251

Para la implementación en cuestión se decidió utilizar un kernel de tamaño 5x5 para las 2 primeras capas convolucionales, y uno 3x3 para las 2 restantes que se definieron al momento de diseñar la red neuronal. Al trabajar con Redes Neuronales Convolucionales es común utilizar un kernel_size de pequeñas dimensiones, debido a la complejidad computacional que trae consigo el generar el output de las capas convolucionales al trabajar con dimensiones de tamaño considerable para el kernel.[61]

---

[61] SAHOO, Sabyasachi. *Deciding optimal kernel size for CNN*. 2018. URL: https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363.

Se utilizó un stride de 2, por lo que el kernel en cada capa convolucional llevó a cabo un producto matricial cada 2 valores (horizontal y verticalmente) con base en la matriz que se recibió como entrada, desplazándose de izquierda a derecha y de arriba hacia abajo (tal y como al leer un texto), y de esa manera se generaba la matriz de salida que era analizada por la siguiente capa de la red.[62]

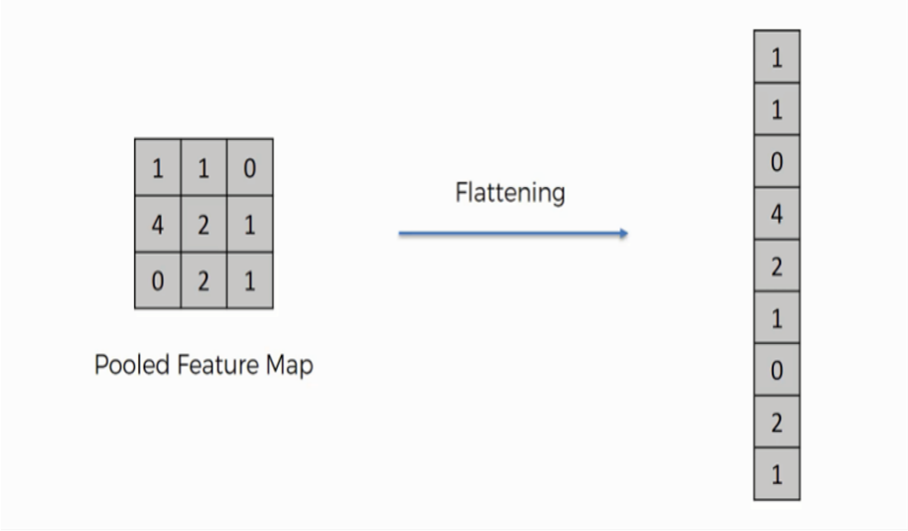
Después de haber definido las capas convolucionales, se utilizó una flatten layer para "alisar" la matriz de salida de la última capa convolucional de la red (tal y como se puede ver en la Figura 39). Con el propósito de que la información que se obtuvo a través de las capas convolucionales sea procesada adecuadamente por las dense layers que componen la estructura final de la CNN (para ello es necesario que como entrada, la primera dense layer de la red reciba un arreglo en 1D).[63]

---

[62] BROWNLEE, Jason. *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks*. 2019. URL: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/.

[63] YAMASHITA, Rikiya, *et al.* "Convolutional neural networks: an overview and application in radiology". En: *Insights into Imaging* 9 (2018), págs. 611 -629.

**Figura 39.** Flattening.




**Fuente:** Convolutional Neural Networks (CNN): Step 3 - Flattening (Consultado el 18 de Julio del 2022) Disponible en: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

**5.1.2. MODELO PROPUESTO.** El modelo propuesto se diseñó con base en la red neuronal expuesta en el repository de github del usuario Chino wuzht.[64] También se adaptó la implementación utilizada en dicho repositorio para entrenar el agente, debido a que el modelo en cuestión fue desarrollado con base en el framework PyTorch, y por ende no se puede utilizar el código al que recurrió el grupo de investigación del MIT para entrenar redes neuronales diseñadas mediante TensorFlow (aun así, ambos códigos son similares en cuanto a su implementación y lógica a seguir se refiere).

---

[64] WUZHT. *Play OpenAI Gym game of Pong using Deep Q-Learning*. 2020. URL: https://github.com/wuzht/DQN_Pong.

El modelo propuesto se construyó a manera de red neuronal convolucional tal y como se puede ver en la Figura 40.

**Figura 40.** Modelo propuesto Pong.

```
class DQN(nn.Module):
    def __init__(self, input_shape, n_actions):
        super(DQN, self).__init__()

        self.conv = nn.Sequential(
            nn.Conv2d(input_shape[0], 16, kernel_size=6, stride=4),
            nn.ReLU(inplace=True),
            nn.Conv2d(16, 32, kernel_size=3, stride=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 32, kernel_size=2, stride=1),
            nn.ReLU(inplace=True)
        )

        conv_out_size = self._get_conv_out(input_shape)
        self.fc = nn.Sequential(
            nn.Linear(conv_out_size, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, n_actions)
        )

    def _get_conv_out(self, shape):
        o = self.conv(torch.zeros(1, *shape))
        return int(np.prod(o.size()))

    def forward(self, x):
        conv_out = self.conv(x).view(x.size()[0], -1)
        return self.fc(conv_out)
```

Al igual que en el modelo de referencia los filtros se definieron de manera ascendente para las capas (a excepción de la última, ya que esta como salida únicamente genera los Q-Values de la red), teniendo en cuenta que en PyTorch se manejan filtros de entrada y de salida para cada capa, por lo que el número de filtros de salida de una capa suele ser el mismo número de filtros de entrada de la siguiente.
El stride y el kernel_size de las capas convolucionales, de manera similar que en el modelo de referencia, se definieron también de manera descendente.

## 5.2. ENTRENAMIENTO DE LOS MODELOS

Para el entrenamiento de los modelos diseñados mediante TensorFlow se optó por recurrir a una gran cantidad de algoritmos, que sirvieron para entrenar los agentes inteligentes que se desenvolvían en el videojuego de Atari 2600 Pong.

**5.2.1. SELECCIONAR ACCIÓN** Para que el agente seleccionara la acción a realizar fue necesario recurrir a la función choose_action que se utilizó para la implementación del Péndulo Invertido (ver Figura 24).

**5.2.2. DISEÑO DE LA MEMORIA DEL AGENTE** Para definir la memoria del agente se decidió hacer uso de la clase Memory que se implementó previamente para el caso de estudio del Péndulo Invertido (ver Figura 25).

**5.2.3. FUNCIÓN DE ENTRENAMIENTO** Esta función es el mismo algoritmo train_step que se utilizó para alterar los pesos de las redes neuronales que fueron usadas para entrenar los agentes inteligentes que lograron superar el Problema del Péndulo Invertido (ver Figura 28).

**5.2.4. CÁLCULO DE LA PÉRDIDA** En este caso es necesario minimizar la pérdida en busca del mayor rendimiento posible al igual que se hizo durante la implementación del Péndulo Invertido, por ello se utilizó también la función compute_loss (ver Figura 27).

**5.2.5. RECOMPENSA CON DESCUENTO** Se recurrió a la función discount_rewards utilizada en el Problema del Péndulo Invertido para calcular las pérdidas acumuladas y con descuento que obtuvieron los agentes inteligentes mientras se desenvolvían en el entorno del videojuego de Atari 2600 Pong, adaptando la función para utilizar

un gamma de 0,99 (mayor que en el caso del Péndulo Invertido e igual que en la implementación que se usó para probar las redes diseñadas mediante PyTorch) y tener en cuenta las recompensas diferentes de 0 sin acumular y sin descuento (debido a que son las de mayor importancia que se pueden llegar a adquirir, se obtienen a medida que el agente o el contrincante va ganando puntos). El algoritmo adaptado puede visualizarse mediante la Figura 41.

**Figura 41.** Recompensa con descuento Pong.

```
[ ]  def normalize(x):
       x -= np.mean(x)
       x /= np.std(x)
       return x.astype(np.float32)
```

```
[ ]  def discount_rewards(rewards, gamma=0.99):
       discounted_rewards = np.zeros_like(rewards)
       R = 0
       for t in reversed(range(0, len(rewards))):

           if rewards[t] != 0:
             R = 0

           R = R * gamma + rewards[t]
           discounted_rewards[t] = R

       return normalize(discounted_rewards)
```

**5.2.6. CONFIGURACIÓN PREVIA AL ENTRENAMIENTO** Tal y como se puede ver en la Figura 42, el learning_rate adquirió un valor de 0,001 (en busca de mejores resultados, se trabajó con 0,0001 para la implementación que se centró en la red diseñada mediante PyTorch), la constante MAX_ITERS fue utilizada para hacer referencia a la cantidad de episodios que se tuvieron en cuenta para el entrenamiento, y también se definió el valor que adquirió el batch_size utilizado (para la implementación en PyTorch este fue de 32).

**Figura 42.** Configuración previa Pong.

```
[ ] learning_rate = 1e-3
    MAX_ITERS = 3
    batch_size = 5


    pong_model = create_pong_model()
    optimizer = tf.keras.optimizers.Adam(learning_rate)
    iteration = 0


    smoothed_reward = mdl.util.LossHistory(smoothing_factor=0.9)
    smoothed_reward.append(0)
    plotter = mdl.util.PeriodicPlotter(sec=15, xlabel='Iterations', ylabel='Win Percentage (%)')


    envs = [create_pong_env() for _ in range(batch_size)]
```

El batch size en el ámbito de las redes neuronales hace referencia a la cantidad de muestras de entrada que son procesadas antes de que los pesos de la red se actualicen. Por ende, dicha modificación ocurre cada batch size estados del entorno

procesados por la red.[65] La constante pong_model se utilizó para instanciar la red neuronal creada, y optimizer sirvió para hacer referencia al optimizador utilizado para el entrenamiento de la red (para las implementaciones que se centraron en el entorno del videojuego de Atari 2600 Pong se utilizó el optimizer Adam).

La variable iteration durante el proceso de entrenamiento fue utilizada para hacer referencia a la cantidad de episodios que se iban teniendo en cuenta a medida que el agente se iba entrenando, y las funciones propias del grupo de investigación del MIT mdl.util.LossHistory y mdl.util.PeriodicPlotter fueron utilizadas para poder visualizar la gráfica que muestra la relación existente entre rendimiento del agente y número de episodios de entrenamiento que se han procesado.

Para la implementación en cuestión fue necesario crear batch_size entornos del videojuego de Atari 2600 Pong, con el propósito de entrenar la red neuronal con agentes desenvolviéndose simultáneamente en dichos entornos, para de esa manera conocer la naturaleza de la mayor cantidad de estados del entorno posibles lo más rápido que se pueda (debido a que el entorno de un videojuego es mucho más complejo que aquel que se utilizó durante la implementación del Problema del Péndulo Invertido).

**5.2.7. ALGORITMO GENERAL** Para entrenar las Deep Q-Networks (creadas mediante TensorFlow) que fueron utilizadas por los agentes que aprendieron a ganarle al contrincante en el videojuego de Atari 2600 Pong fue necesario recurrir al algoritmo general expuesto en la Figura 43.

Para que el agente o el contrincante ganen una partida es necesario anotar 21 puntos. El while que se tiene en cuenta en el algoritmo general fue utilizado para

---

[65] BROWNLEE, Jason. *Difference Between a Batch and an Epoch in a Neural Network*. 2022. URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20batch%20size%20is%20a%20number%20of%20samples%20processed%20before,samples%20in%20the%20training%20dataset..

**Figura 43.** Algoritmo general Pong.

```
[ ]  games_to_win_episode = 21


     while iteration < MAX_ITERS:

       plotter.plot(smoothed_reward.get())

       tic = time.time()



       memories = mdl.lab3.parallelized_collect_rollout(batch_size, envs, pong_model, choose_action)
       print(time.time()-tic)
       print("Episodio ", iteration)

       batch_memory = aggregate_memories(memories)


       total_wins = sum(np.array(batch_memory.rewards) == 1)
       total_games = sum(np.abs(np.array(batch_memory.rewards)))
       win_rate = total_wins / total_games
       smoothed_reward.append(100 * win_rate)


       train_step(
           pong_model,
           optimizer,
           observations = np.stack(batch_memory.observations, 0),
           actions = np.array(batch_memory.actions),
           discounted_rewards = discount_rewards(batch_memory.rewards)
       )

       if iteration == MAX_ITERS-1:

         pong_model.save("/content/drive/My Drive/Deep Reinforcement Learning/Agentes entrenados/1000epspong.h5")

       iteration += 1
```

que el entrenamiento se lleve a cabo durante MAX_ITERS episodios. La función propia del grupo de investigación del MIT mdl.lab3.parallelized_collect_rollout fue utilizada para que con base en la Deep Q-Network que se está entrenando, agentes inteligentes se desenvuelvan simultáneamente en los entornos en paralelo que se crearon del videojuego de Atari 2600 Pong, y se almacenen en memoria los estados del entorno alcanzados por los agentes, las acciones que llevaron a cabo en estos y las recompensas que se obtuvieron como consecuencia de ello.

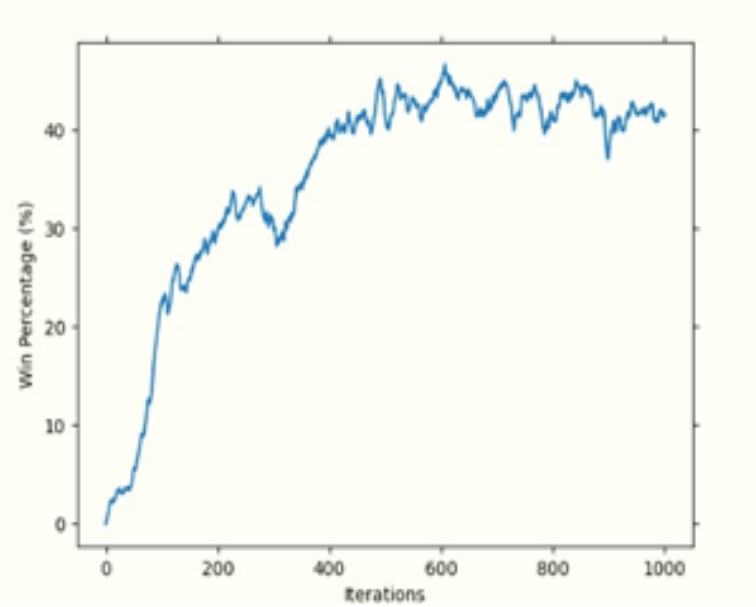
La información almacenada en memoria es utilizada para ir calculando el porcentaje de victorias (puntos obtenidos por el agente/puntos obtenidos en conjunto por

el agente y el contrincante) a medida que se va llevando a cabo el proceso de entrenamiento, dicha información también es utilizada por la función train_step para ir actualizando los pesos de la Deep Q-Network que se está entrenando.
El condicional if que es utilizado cuando acaba de terminar el último episodio de entrenamiento sirve para guardar los pesos que ha adquirido la red tras haberse entrenado por MAX_ITERS episodios. Debido al extenso tiempo que le puede tomar a los agentes entrenarse para superar al contrincante en el videojuego de Atari 2600 Pong, fue necesario en reiteradas ocasiones guardar y cargar los pesos a medida que se iban entrenando los modelos, debido a que era obligatorio apagar la máquina utilizada para llevar a cabo el entrenamiento al final de cada día que se utilizó.

## 5.3. COMPARAR DESEMPEÑO DE LOS AGENTES

A raíz de los resultados obtenidos se consideró necesario comparar el desempeño de los agentes mediante gráficas y videos en los que se pueden visualizar las partidas que se llevaron a cabo entre el agente entrenado y el contrincante en el videojuego de Atari 2600 Pong para cada modelo que se entrenó.
En primer lugar, es necesario visualizar la gráfica del desempeño del agente que se entrenó con base en la Deep Q-Network diseñada mediante TensorFlow propuesta por el grupo de investigación del MIT (ver Figura 44).

**Figura 44.** Resultados Pong modelo de referencia.




Después de haber entrenado el modelo de referencia durante 1000 episodios, el desempeño del agente se estancó en un 40 % de victorias, por lo que en cada ronda de una partida existía un 40 % de probabilidad de que la inteligencia artificial entrenada fuera la que anotara un punto, lo cual no es del todo beneficioso ya que el 60 % de las veces restantes ganará el contrincante, pero aún así los resultados obtenidos pueden considerarse parcialmente aceptables.

El video que se generó del agente entrenado mediante el modelo de referencia enfrentándose al contrincante en el videojuego de Atari 2600 Pong[66] deja en evidencia que el agente aprendió a defenderse lo más que se pudo, llegando a impedir que el contrincante anotara puntos en más de una ocasión, pero aún así el resultado final de la partida fue de 21 contra 9, perdiendo el agente. A pesar de que aprendió a buscar el proyectil cuando este se acerca para impedir que el contrincante anote un

---

[66] RODRÍGUEZ, Julián. *Pong Modelo Referencia*. 2022. URL: https://www.youtube.com/shorts/gPdclEnfQcU.

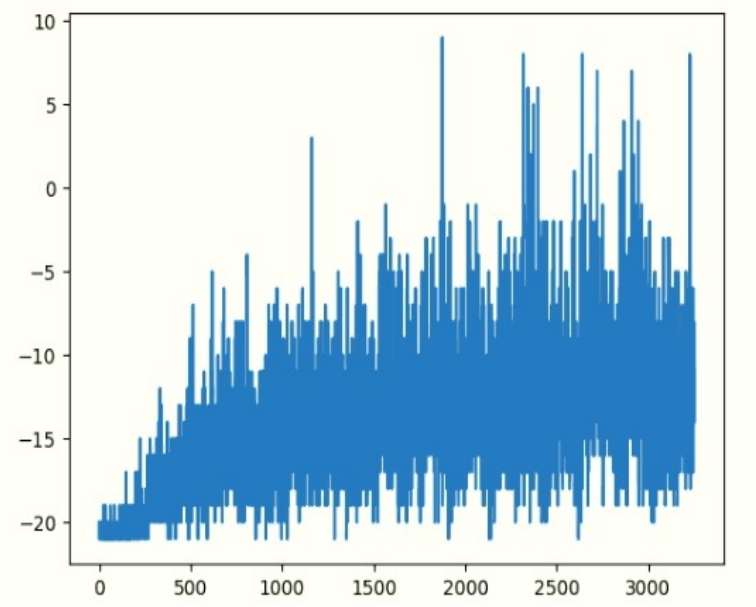
punto, la precisión del agente aún deja mucho que desear y por ende la mayoría de las veces se suele perder.

Para obtener mejores resultados se decidió utilizar el modelo propuesto. Debido a que la gráfica que representaba el porcentaje de victorias del agente que se entrenó mediante el modelo de referencia era un poco confusa, para la implementación que se tuvo en cuenta en este caso se decidió utilizar una gráfica más intuitiva para poder visualizar el desempeño del agente, en la que el eje x representa el número de episodios que se iban teniendo en cuenta durante el proceso de entrenamiento, y el eje y hace referencia a un valor entre 21 y -21 que es igual al puntaje obtenido por el agente menos el puntaje obtenido por el contrincante en cada partida.

El modelo propuesto se entrenó durante 20000 episodios, debido a lo extenso del entrenamiento fue necesario llevarlo a cabo mediante una gran cantidad de intervalos. Para poder visualizar la mejoría que se presentaba a medida que el agente iba entrenándose cada vez más se van a comparar los resultados que se obtuvieron al comienzo del entrenamiento, con aquellos que se alcanzaron al final.

El primer intervalo que se tuvo en cuenta para entrenar al agente estuvo compuesto de 3250 episodios de entrenamiento. Tal y como se puede ver en la Figura 45, el desempeño del agente deja mucho que desear al comienzo del proceso de entrenamiento, debido a que la mayoría de las veces el contrincante ganaba la partida (ya que la media de los puntajes expuestos en la gráfica tiende a adquirir valores negativos, en otras palabras, el contrincante la mayoría de las veces adquiría un mayor puntaje que el agente entrenado).

**Figura 45.** Resultados Pong modelo propuesto primer intervalo.



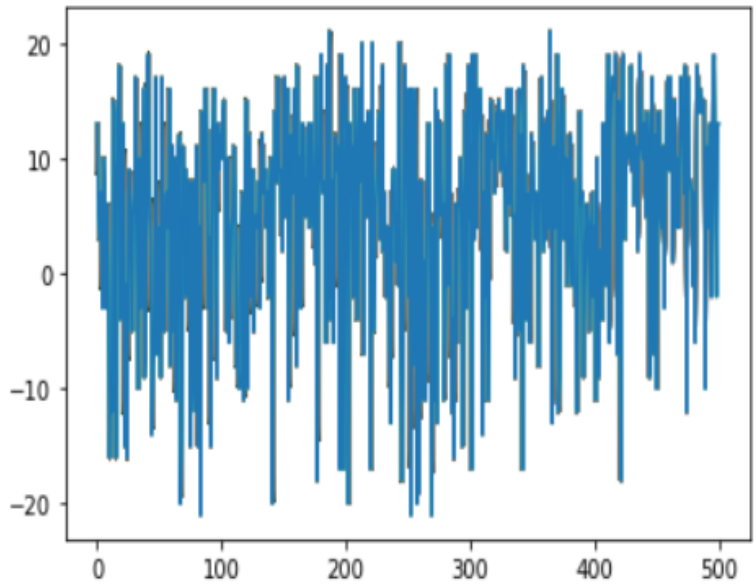

El video que se generó tras haber entrenado al agente durante 3250 episodios[67] deja en evidencia que el agente logró demostrar un mejor desempeño que tras haberse entrenado por 1000 episodios con base en el modelo de referencia, el puntaje final de la partida fue de 21 contra 12, ganando el contrincante, demostrando de esa manera lo acertada que resulta ser la Figura 45.

El entrenamiento siguió llevándose a cabo, hasta que al final se registró el último intervalo que se tuvo en cuenta para entrenar al agente con base en el modelo propuesto, el intervalo final hizo referencia a los últimos 500 episodios del entrenamiento. Para ver el desempeño final del agente se recurre a la Figura 46.

---

[67] RODRÍGUEZ, Julián. *Pong Modelo Propuesto primer intervalo*. 2022. URL: https://youtube.com/shorts/o-wspmPKVt4.

**Figura 46.** Resultados Pong modelo propuesto último intervalo.




La gráfica muestra que los resultados siguieron variando en gran medida hasta finalizar el proceso de entrenamiento, con la notoria diferencia de que, a comparación del primer intervalo, en este caso la media de los puntajes adquiere valores positivos y cercanos a cero, por lo que la mayoría de las veces el agente entrenado logró ganarle al contrincante o perdió por poco.

El video que se generó tras terminar de entrenar al agente inteligente[68] demuestra que la gráfica es acertada, debido a que logró ganarle al contrincante 21 contra 4. Visualizando el video es posible percibir que el agente en más de una ocasión se centró en explotar las mejores acciones a realizar posibles, debido a que aprendió a mantenerse estático en un lugar en específico del entorno, en el que por defecto el proyectil llegará la mayoría de las veces si inició la ronda saliendo por la izquierda. El agente también aprendió a golpear el proyectil ocasionando que este rebote en

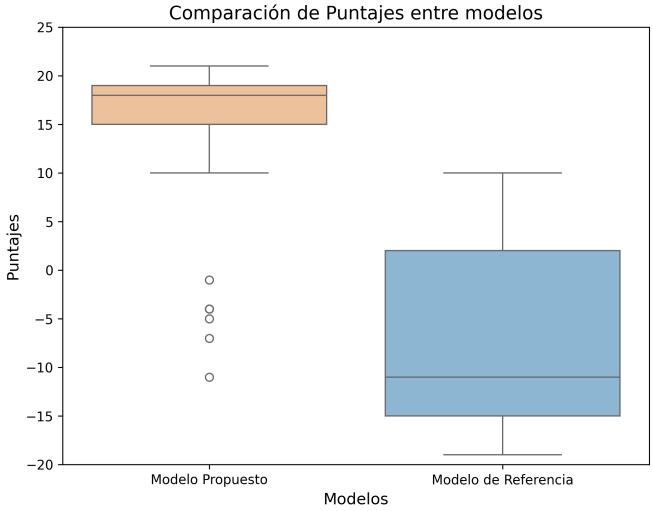
---

[68] RODRÍGUEZ, Julián. *Pong Modelo Propuesto último intervalo*. 2022. URL: https://youtube.com/shorts/jVsvRqywv68?feature=share.

la esquina inferior del entorno de juego para que al contrincante se le dificulte más evitar que se anote un punto. Incluso, en caso de que el agente pierda el control del proyectil, aprendió a volver a estabilizarlo para anotar puntos la mayoría de las veces.

A pesar de que visualizando la Figura 46 se puede llegar a considerar que los resultados aún no son del todo buenos, analizando el rendimiento de los modelos tras llevar a cabo 100 partidas (ver Figura 47), se puede corroborar que la eficiencia del modelo propuesto es mucho mejor que la del modelo de referencia.

**Figura 47.** Comparación de Puntajes entre modelos (Pong).




Por lo que en promedio el agente inteligente aprendió a ganarle al contrincante en el videojuego de Atari 2600 Pong con una ventaja de 14 a 15 puntos, entrenándose con base en el modelo propuesto durante 20000 episodios.

# 6. DEEP RECURRENT ATTENTION MODEL PARA IDENTIFICAR REGIONES DE INTERÉS CLÍNICO EN IMÁGENES HISTOLÓGICAS

El avance de las nuevas tecnologías ha revolucionado el campo de la medicina, permitiendo un análisis más rápido y detallado de las imágenes patológicas mediante el uso de herramientas basadas en aprendizaje de máquina. Las Redes Neuronales Convolucionales, las Redes Generativas Adversarias y el Aprendizaje por Refuerzo son algunas de las técnicas de Aprendizaje Profundo utilizadas para procesar y clasificar imágenes histopatológicas. En este capítulo nos centraremos en la utilización de RNN, para mediante el Aprendizaje por Refuerzo y una política de Atención Visual identificar las regiones de mayor relevancia clínica en las imágenes, emulando las estrategias de un patólogo experto. En este estudio evaluamos estas técnicas para confirmar que el aprendizaje por refuerzo en este escenario resalta regiones de interés similares a las identificadas por un experto .

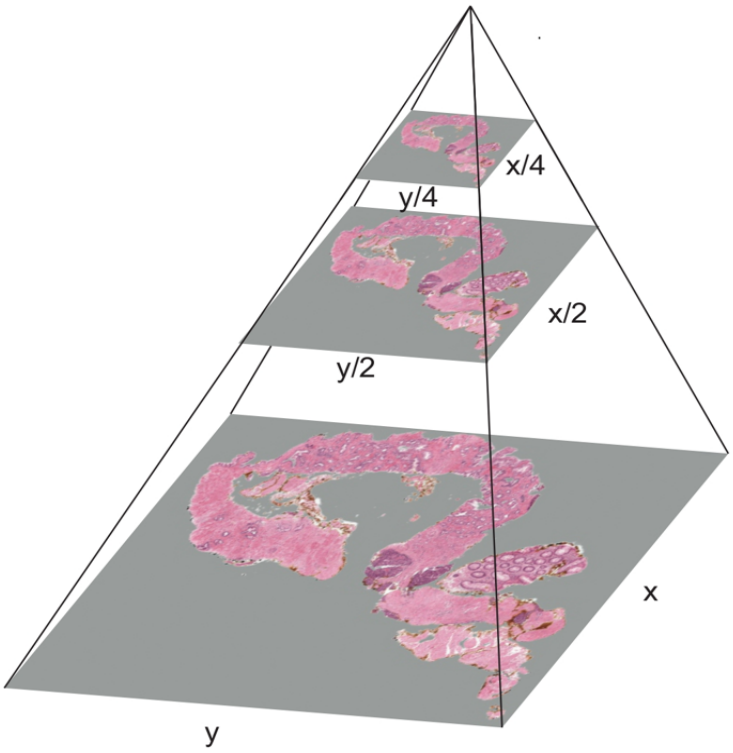
## 6.1. MATERIALES

Las imágenes histológicas mediante las cuales se entrenó el modelo se obtuvieron del Prostate cANcer graDe Assessment (PANDA) Challenge[69], un concurso virtual de Deep Learning en Kaggle, que ofrece un dataset de 10616 imágenes histológicas de próstata en formato .tiff, cada una con un máximo de 3 niveles de resolución. Este dataset se dividió en tres conjuntos diferentes: entrenamiento, validación y evaluación. Estos tres conjuntos se definen de la siguiente manera: 90 % para entrenamiento, 5 % para validación y 5 % para la evaluación. Con 531 imágenes re-

[69] BULTEN, W., *et al.* "Artificial intelligence for diagnosis and Gleason grading of prostate cancer: the PANDA challenge". En: *Nature Medicine* 28 (2022), págs. 154-163. DOI: 10.1038/s41591-021-01620-2.

presentando el 5 % del total (validación), 9554 imágenes corresponden al 90 % y 531 imágenes corresponden al 5 % (evaluación).

**Figura 48.** Representación piramidal de las imágenes histológicas. Las imágenes se codifican en diferentes resoluciones, que también están asociadas a diferentes niveles de aumento. Esta representación facilita el acceso a diferentes regiones de la imagen a diferentes aumentos.




**Fuente:** Elaboración propia

Junto al dataset se encuentra la información más importante a tener en cuenta para que el entrenamiento se lleve a cabo de manera adecuada. En dicha información se encuentra el id de cada imagen, junto a la respectiva institución médica en la que fue adquirida (Institución Karolinska o Radboud), y el Isup Grade y el Gleason Score de cada tejido (sistemas de medición utilizados para identificar la agresividad de un cáncer de próstata).

Para resolver adecuadamente el problema de clasificación para la implementación

del modelo, se decidió analizar el Gleason Score de cada imagen histológica. Las imágenes que carecían de valor alguno para este parámetro fueron etiquetadas con un valor de "1" (libre de cáncer), mientras que las imágenes que tenían un Gleason Score fueron etiquetadas con un valor de "0" (tejidos cancerosos). En total, 2893 imágenes corresponden a tejido libre de cáncer, y 7723 corresponden a imágenes con tejido canceroso

**Cuadro 15.** ISUP Grade y Gleason Score

| ISUP grade groups | Gleason Score | Características |
|---|---|---|
| Grade group 1 | 6 | Menos agresivo<br>Velocidad de crecimiento muy lenta<br>Bajo riesgo |
| Grade group 2 | 3 + 4 = 7 | Ligeramente agresivo<br>Velocidad de crecimiento lenta<br>Bajo/intermedio riesgo |
| Grade group 3 | 4 + 3 = 7 | Moderadamente agresivo<br>Velocidad de crecimiento rápida<br>Intermedio/alto riesgo |
| Grade group 4 | 4 + 4 = 8, 3 + 5 = 8,<br>5 + 3 = 8 | Agresivo<br>Velocidad de crecimiento muy rápida<br>Alto riesgo |
| Grade group 5 | 4 + 5 = 9, 5 + 4 = 9<br>5 + 5 = 10 | Muy agresivo<br>Velocidad de crecimiento muy rápida<br>Alto riesgo |
| N/A | N/A | Células sanas<br>Tejido sin cáncer |

Debido a que las imágenes .tiff ocupan una gran cantidad de espacio en memoria y con el propósito de reducir la complejidad computacional lo más que se pueda, se decidió trabajar con parches que se extrajeron de las imágenes .tiff, en vez de trabajar con la totalidad de las imágenes como tal.

El proceso realizado para obtener los parches de las imágenes histológicas se llevó a cabo con base en la implementación expuesta en el notebook de Kaggle [PANDA]

Optimized tiling (+ tf.data.Dataset)[70]. Mediante dicho código la mayor cantidad de parches que se podían extraer de las áreas con tejido en las imágenes histológicas se obtuvieron, de tal manera que no se almacenaron parches en blanco, debido a que estos interferirían con el adecuado proceso de entrenamiento del modelo a realizar (es necesario tener en cuenta únicamente información relevante).

Para la extracción de los parches en primer lugar se dividió la imágen histológica en N parches de igual tamaño dependiendo de las dimensiones definidas, posteriormente se almacenó la información presente en las máscaras. Únicamente los parches que poseían un porcentaje significativo de máscara en ellos, fueron utilizados para entrenar el modelo en cuestión.

La máscara de una imagen histológica se refiere al área de la imagen que contiene tejido para un análisis posterior. Se utiliza principalmente en procedimientos de clasificación y procesamiento de imágenes para reducir el ruido y la complejidad computacional al analizar las imágenes. Esto se debe a que es mucho más sencillo llevar a cabo este proceso basándose en imágenes bicromáticas que se apoyan únicamente en regiones con una máscara (áreas relevantes) y sin una máscara, en lugar de trabajar con imágenes que involucran múltiples niveles de color.

## 6.2. HISTODRAM

El modelo utilizado para nuestra implementación, que fue entrenado con base en parches extraídos de imágenes histológicas, es un Deep Recurrent Attention Model, tal y como se expone en el artículo Deep Recurrent Attention Models for Histopathological Image Analysis [30].

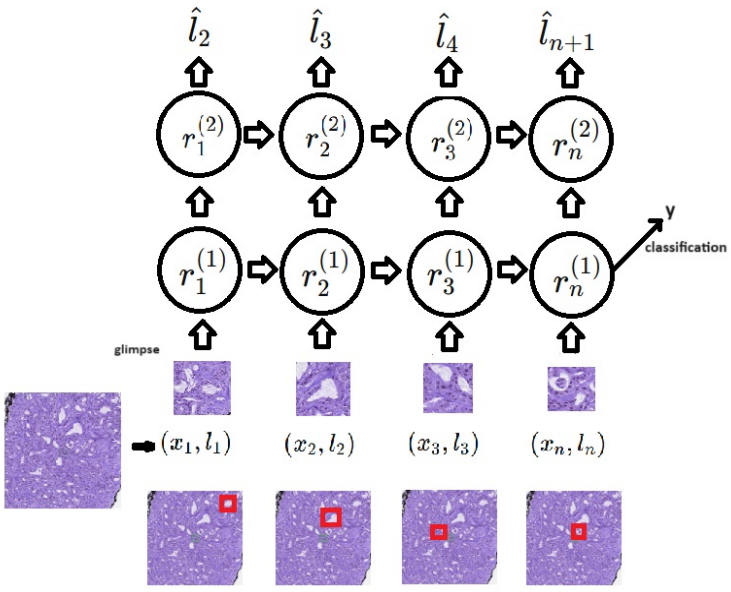
El DRAM utiliza un gran número de redes neuronales, que en conjunto generan in-

---

[70] AKENSERT. *PANDA Optimized Tiling & TF Data Dataset*. `https://www.kaggle.com/code/akensert/panda-optimized-tiling-tf-data-dataset`. Accessed: 2024-07-21. 2022.

formación que es procesada secuencialmente, utilizando una RNN y una política de Atención Visual al mismo tiempo. De esta manera, el modelo aprende a enfocar su atención principalmente en las áreas de tejido más relevantes para clasificar imágenes histológicas de manera más sencilla.

**Figura 49.** Deep Recurrent Attention Model.




**Adaptado de:** Deep Recurrent Attention Models for Histopathological Image Analysis.

Dado que el tamaño de los parches puede ser demasiado grande para que el modelo se entrene fácilmente con ellos, se decidió trabajar con glimpses extraídos de los parches generados para entrenar el modelo.

La primera red que compone el DRAM es la Glimpse Network. Es responsable de recibir en cada step de entrenamiento t una ubicación $l_t$, que es la ubicación exacta de donde se extrajo el glimpse en el parche. $l_t$ se inicializa en el primer step en (0,0), de modo que el primer glimpse analizado por el DRAM se extrae del centro de la

imagen. La red también recibe el glimpse $x_t$ que fue extraído en formato de imagen. Posteriormente, $x_t$ se envía a través de capas convolucionales, se le aplica Max pooling y finalmente, a través de una Fully Connected Layer, se obtiene un vector $g_{x_t}$ , que representa 'qué' se está analizando. De manera similar, la ubicación $l_t$ pasa a través de Fully Connected Layers, obteniéndose así el vector $g_{l_t}$, que representa 'dónde' se encuentra lo que se está analizando. Finalmente, es necesario unificar el 'qué' y el 'dónde' en un único vector $g_t$. Para ello, se realiza una multiplicación elemento a elemento entre los vectores $g_{x_t}$ y $g_{l_t}$, y el vector resultante es la salida de la red.

Luego, se utiliza la Core Network, una Recurrent Neural Network compuesta por dos unidades LSTM que reciben el vector $g_t$ y así se obtienen los feature vectors $r_t^{(1)}$ y $r_t^{(2)}$ (solo durante el primer step se recibe únicamente el vector $g_t$ porque luego, al trabajar con una RNN, también se tiene en cuenta la información almacenada en los feature vectors de los steps anteriores para el entrenamiento). Se utilizaron unidades LSTM porque son óptimas para que un modelo aprenda dependencias de largo alcance y por su estabilidad en las dinámicas del aprendizaje profundo.

El feature vector $r_t^{(2)}$ se utiliza como parámetro de entrada por la Location Network, una red neuronal compuesta por Fully Connected Layers que produce como salida la ubicación del siguiente glimpse que debe ser extraído de la imagen analizada por el modelo. El objetivo del desarrollo de este proyecto es entrenar esta Location Network de la manera más óptima posible, de modo que, dependiendo de la imagen, el modelo genere la ubicación más óptima para extraer un glimpse con la información más relevante de la imagen, con el fin de resolver el problema de clasificación más fácilmente.

Para entrenar la Location Network es necesario utilizar Aprendizaje por Refuerzo. Para implementarlo se requiere de una recompensa o penalización que depende del desempeño del agente. Para obtener recompensas y penalizaciones se utiliza

la Classification Network. Esta recibe el feature vector $r_t^{(1)}$ y genera una predicción del tipo de tejido histológico analizado por el modelo (si es tejido sano o canceroso). Dependiendo de si la Classification Network acierta o no en su predicción, se obtiene una recompensa o penalización y, según esta información, se evita obtener glimpses de ciertas áreas de las imágenes analizadas y, por el contrario, se motiva al agente a extraer glimpses de áreas de mayor relevancia clínica para llevar a cabo la clasificación de la mejor manera posible.

Para llevar a cabo la implementación se utilizaron una gran cantidad de hiperparámetros. `val_size` y `test_size` se usaron para referirse a los porcentajes de los conjuntos de validación y evaluación respectivamente, los cuales se obtuvieron con base en el número total de imágenes presentes en el dataset utilizado para entrenar el modelo.

`num_glimpses` representa el número de glimpses que debían extraerse de cada parche analizado por el modelo. Se utilizaron valores distintos para el Batch Size, siendo estos `batch_size`, `eval_batch_size` y `test_batch_size` para el entrenamiento, la validación y la evaluación, respectivamente. El Batch Size representa el número de parches que fueron analizados simultáneamente por el DRAM durante cada Step del proceso de entrenamiento.

`input_shape` y `patch_size` toman los mismos valores, porque las dimensiones de los parches analizados por el modelo corresponden a las mismas dimensiones de entrada para este.

Además, se definió el `glimpse_size`, que se refiere a las dimensiones de los glimpses que se extrajeron de los parches analizados por el modelo durante el proceso de entrenamiento.

**6.2.1. ENTRENAMIENTO DEL MODELO** Durante el entrenamiento del modelo, es fundamental evaluar su rendimiento para garantizar su eficacia en la clasificación.

Para comparar los resultados obtenidos después de entrenar el modelo, se utilizaron dos métricas para evaluar la eficiencia del modelo implementado en la resolución del problema de clasificación.

En primer lugar, se empleó la curva ROC, que representa gráficamente la relación que existe entre la Tasa de Verdaderos Positivos (TPR) y la Tasa de Falsos Positivos (FPR) durante el proceso de evaluación.

Además, se analizó el Área Bajo la Curva (AUC), que corresponde al área bajo la curva ROC. Cuanto mayor sea el AUC, más eficiente es el modelo.

La siguiente tabla permite visualizar los valores de AUC obtenidos tras llevar a cabo la implementación DRAM cambiando una gran cantidad de hiperparámetros, tal y como se muestra a continuación.

| Experiment | Batch size | Resolution | Glimpse size | Num Glimpses | Epochs | Learning Rate start | Learning Rate min | AUC ROC |
|---|---|---|---|---|---|---|---|---|
| Exp 1 | 8 | 512 level 1 | 32 | 12 | 10000 | 0.001 | 0.0001 | 0.63 |
| Exp 2 | 16 | 1536 level 0 | 96 | 12 | 5000 | 0.001 | 0.0001 | 0.73 |
| Exp 3 | 32 | 512 level 1 | 32 | 12 | 1000 | 0.001 | 0.0001 | 0.67 |
| Exp 4 | 32 | 1536 level 0 | 96 | 6 | 5000 | 0.001 | 0.0001 | 0.66 |
| Exp 5 | 8 | 2560 level 0 | 160 | 6 | 2000 | 0.001 | 0.0001 | **0.82** |
| Exp 6 | 16 | 1536 level 0 | 96 | 6 | 5000 | 0.01 | 0.0001 | 0.50 |
| Exp 7 | 16 | 1536 level 0 | 96 | 6 | 5000 | 0.001 | 0.0001 | 0.53 |
| Exp 8 | 16 | 1536 level 1 | 96 | 6 | 1000 | 0.001 | 0.0001 | 0.75 |

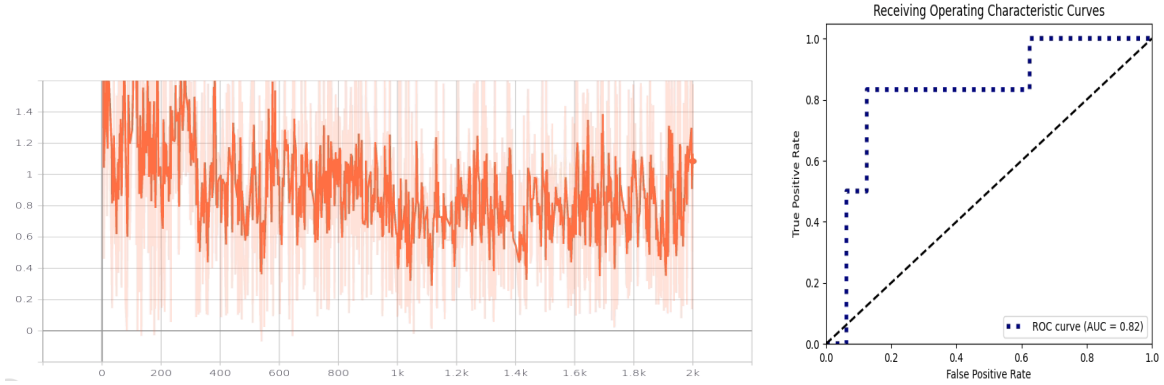
**Cuadro 16.** DRAM Implementation experiments.

Los resultados obtenidos para el mejor experimento nos permiten confirmar que el agente inteligente es capaz de clasificar correctamente los tejidos histológicos el 82 % del tiempo. Se llevaron a cabo una gran cantidad de experimentos ajustando número de glimpses, batch size, dimensiones de los parches y nivel de resolución de los mismos, learning rate y demás, y se escogió el mejor para análisis de resultados y métricas.

De manera similar, se puede visualizar el gráfico resultante obtenido al final del proceso de entrenamiento del mejor experimento. Este gráfico se utiliza para relacionar la pérdida que se buscó disminuir lo máximo posible (hybrid loss) con los Steps de entrenamiento empleados para entrenar al agente inteligente (2000 Steps, en esta

implementación).

**Figura 50.** Resultados DRAM - La imagen superior es el gráfico que representa la relación entre la Hybrid Loss y los Steps durante el proceso de entrenamiento y validación. La imagen inferior muestra la curva ROC junto con el área bajo la curva (AUC), lo cual es útil para identificar qué tan bien está entrenado el agente para el problema de clasificación.




Al analizar el gráfico, se puede confirmar que la pérdida tiende hacia cero, lo que indica que el proceso de entrenamiento se llevó a cabo de manera adecuada. Si se emplea un mayor número de Steps de entrenamiento, es posible que se logren resultados aún mejores.

## 6.3. EVALUACIÓN DE MAPAS DE DENSIDAD DE PROBABILIDAD

Para analizar la precisión del modelo al enfocar tejidos histológicos relevantes, utilizamos mapas de densidad de probabilidad. Estos se diseñaron basándose en Gaussian heatmaps, de manera que la intensidad aumenta a medida que los heat points (que representan a los glimpses predichos por el modelo) se superponen entre sí.
Para diseñar los Gaussian heatmaps, se tomaron en cuenta las locations de los glimpses predichos por el modelo. Los heat points de estos glimpses se dibujaron sobre los parches correspondientes, que luego se posicionaron sobre el tejido histológico para visualizar la imagen completa con los glimpses predichos por el modelo

sobre esta.

Posteriormente, con los heat points obtenidos, se aplicó un filtro Gaussiano con un tamaño de kernel de 1001x1001 y una desviación estándar de 350, junto con un sigma de 30, para realizar el proceso de suavizado en la imagen.

Por otro lado, el muestreo uniforme en la región de solo tejido se tendrá en cuenta para comparar la eficiencia de las predicciones del modelo (donde los glimpses se extraen del tejido) con el análisis de la totalidad del tejido sin aplicar ninguna política de Atención Visual.

Para comparar los resultados, se utilizaron una gran cantidad de métricas. La divergencia KL (divergencia de Kullback–Leibler) se utilizó para identificar el número de bits necesarios para transformar una distribución en otra[71]. En este contexto, cuanto más cerca esté la divergencia KL de cero, mayor será la precisión del modelo, ya que esto indica que un mayor número de glimpses predichos por el modelo están correctamente ubicados en tejidos cancerosos. La divergencia KL se calcula utilizando la siguiente fórmula:

$$KL(P\|Q) = \sum_{x} P(X) * log(\frac{P(x)}{Q(x)}).$$

Una métrica similar que depende en gran medida de la divergencia KL, viene a ser la divergencia JS (divergencia de Jensen–Shannon), la cual también se utilizó para la implementación. Esta métrica también se busca que adquiera un valor cercano a cero, ya que mide la entropía relativa o la diferencia en la información representada por dos distribuciones[72] (tejido canceroso y glimpses predichos por el modelo). La fórmula utilizada para calcular la divergencia JS es la siguiente:

---

[71] ENCORD. *KL Divergence in Machine Learning*. Accessed: 2024-07-21. 2023.

[72] SCIENCE, Towards Data. *How to Understand and Use Jensen-Shannon Divergence*. Accessed: 2024-07-21. 2023.

$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}\left(p\middle\|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\middle\|\frac{p+q}{2}\right)$$

De manera similar, la Mutual Information se utiliza para medir la relación que existe entre dos variables[73]. Cuanto mayor sea este valor, mayor será la relación entre los glimpses predichos por el modelo y el tejido canceroso, indicando así un mejor rendimiento del DRAM. La fórmula necesaria para calcular la Mutual Information es la siguiente:

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(x,y)\log\frac{P(x,y)}{P(x)P(y)}$$

En el contexto de la evaluación de mapas de densidad de probabilidad, se realizó un análisis exhaustivo utilizando parches de imágenes con dimensiones de 2560x2560 y nivel de resolución 0. Este análisis incluyó tanto las máscaras de imagen, que reflejan los glimpses predichos por el modelo (implementación DRAM), como la imagen completa del tejido (muestreo uniforme en el tejido). La siguiente tabla presenta las diversas métricas calculadas durante este proceso, proporcionando una visión detallada del rendimiento del modelo en la evaluación de los mapas de densidad de probabilidad.

**Cuadro 17.** Promedio y Desviación Estándar (entre paréntesis) de Métricas (Divergencia KL, Divergencia JS, y Mutual Information) calculadas durante el proceso de evaluación usando DRAM (Tejido canceroso) y Muestreo Uniforme (Totalidad del tejido)

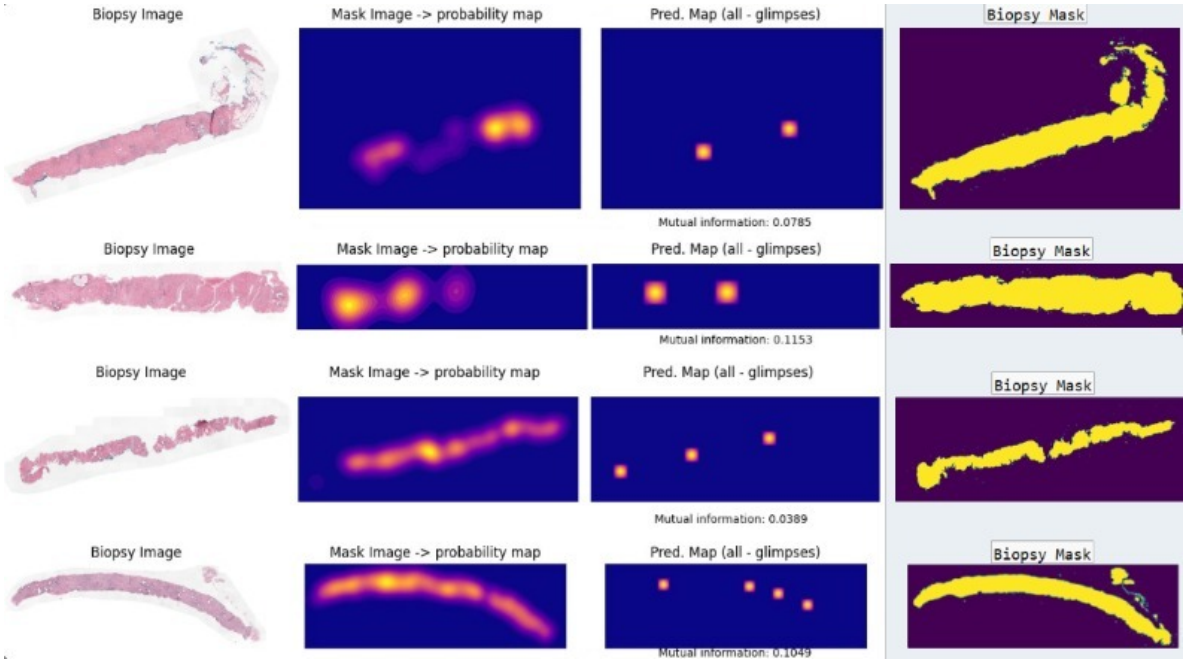
| Métrica / Método | Divergencia KL | Divergencia JS | Mutual Information |
|---|---|---|---|
| Modelo DRAM | 17.80 (11.50) | 0.81 (0.09) | 0.04 (0.03) |
| Muestreo Uniforme* | 1.95 (1.64) | 0.46 (0.24) | 0.097 (0.056) |

[73] VOORHEES, E. M. *Mutual Information*. Accessed: 2024-07-21. 2023.

Los resultados obtenidos corroboran que en promedio, para la mayoría de las imágenes mediante las cuales se calcularon las métricas, el rendimiento del modelo DRAM es menor al del Muestreo Uniforme (simplemente analizar la imagen en su totalidad, sin recurrir a alguna política de atención visual).

Al completar la implementación del DRAM, se obtiene una figura similar a la que se presenta a continuación.

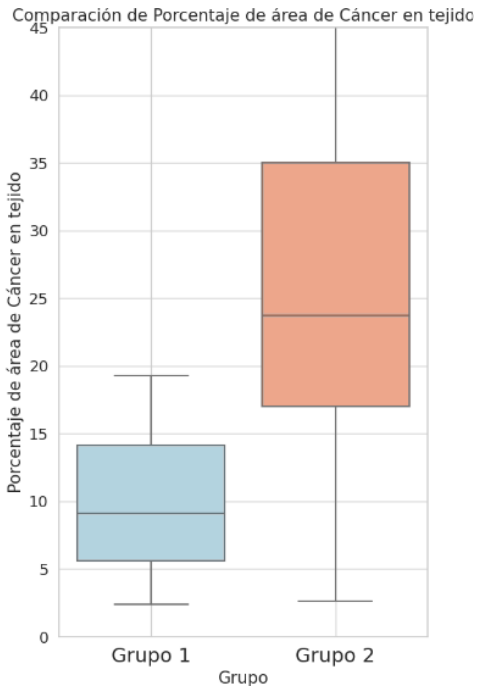
**Figura 51.** Resultados cualitativos para los mapas de densidad de la evaluación. Primera columna: Imagen de baja resolución del tejido histológico, Segunda columna: Máscara suavizada del tejido canceroso de la imagen histológica, Tercera columna: Mapa de calor de los glimpses predichos en cada parche de la imagen, Cuarta columna: Máscara del tejido de la biopsia a tener en cuenta en su totalidad cuando la imagen se analiza mediante Muestreo Uniforme.




La figura en cuestión sirve para visualizar los mapas de densidad de probabilidad de 4 imágenes histológicas, diseñados con base en un modelo DRAM entrenado tras haber llevado a cabo el mejor experimento. Las 2 imágenes superiores corres-

ponden a mapas de densidad de probabilidad en los que el modelo DRAM obtuvo mejores métricas que al analizar la imagen mediante el Muestreo Uniforme. Por el contrario, las 2 imágenes inferiores muestran los mapas de probabilidad en los que el rendimiento del modelo DRAM no es tan bueno como el del Muestreo Uniforme.
Analizando los mapas de densidad de probabilidad, se puede afirmar que la eficiencia del modelo mejora en aquellas imágenes histológicas en las que las anotaciones de la máscara (tejido canceroso) son más detalladas, y de menor tamaño en relación a la totalidad del tejido.
Para corroborar nuestra hipótesis se realizó un boxplot teniendo en cuenta las imágenes mediante las cuales se obtuvieron las métricas del mejor experimento.

**Figura 52.** Comparación de Porcentaje de área de Cáncer en tejido. Grupo 1:El rendimiento del modelo DRAM superó a la distribución uniforme. Grupo 2: El rendimiento de la distribución uniforme superó al modelo DRAM




La figura nos permite corroborar que evidentemente el desempeño del DRAM es mejor en aquellas imágenes histológicas en las que la máscara se encuentra defi-

nida de manera detallada en porciones de tejido canceroso de pequeño tamaño a comparación de la totalidad del tejido. Tiene sentido, debido a que es más eficiente recurrir a una política de Atención Visual en casos en los que hay que centrar la atención no en la totalidad de la imagen, sino únicamente en pequeñas porciones de tejido relevante en esta.

## 6.4. OBSERVACIONES FINALES

Después de calcular las métricas correspondientes y llevar a cabo la implementación adecuadamente, se concluyó que el modelo DRAM es en gran medida capaz de aprender a clasificar correctamente los tejidos histológicos dependiendo de si contienen células cancerosas o no. Este hallazgo se ve reforzado por el hecho de que las métricas indican que el modelo puede mejorar aún más si se entrena durante un período de tiempo más largo, ya que la pérdida tiende a cero durante el entrenamiento. A pesar de que el problema de clasificación se resuelve de manera adecuada la mayoría del tiempo, no se logra visualizar que los glimpses predichos por el modelo correspondan a las denotaciones del patólogo, en las muestras que tienen una zona extensa de tejido canceroso en relación a la totalidad del tejido.
Se logró corroborar que el valor de las métricas aumenta significativamente cuando se trabaja con el modelo DRAM para analizar imágenes histológicas con máscaras detalladas y de pequeño tamaño, en comparación con la totalidad del tejido. Por lo tanto, se puede afirmar que el modelo DRAM funciona incluso mejor que el Muestreo Uniforme para estas imágenes, donde no es necesario centrar la atención en todo el tejido, sino en pequeñas áreas relevantes. Aunque las regiones anotadas suelen ser bastante extensas, logramos identificar que, en situaciones donde las anotaciones son más precisas, existe una correspondencia significativa. Esto resalta la capacidad del modelo para centrarse en áreas específicas que son realmente importantes para la clasificación.

# 7. CONCLUSIONES Y PERSPECTIVAS

Tras haber terminado de llevar a cabo las implementaciones, se evidenció que el Aprendizaje por Refuerzo es eficaz al momento de entrenar agentes inteligentes que aprenden a partir de los errores que cometen. Para simulaciones sencillas, tal y como viene a ser el caso del Problema del Péndulo Invertido, basta con utilizar una Red Neuronal compuesta únicamente por Fully Connected Layers para que el agente aprenda a mantener estable el péndulo lo más que se pueda.

Para implementaciones más complejas, como aquella en la que se simula el comportamiento del videojuego Pong para Atari 2600, se recurrió a Aprendizaje por Refuerzo mediante el uso de Redes Neuronales Convolucionales, debido a que era necesario analizar las imágenes del videojuego constantemente, para que el agente aprendiera a relacionar los movimientos del contrincante y del proyectil para evitar penalizaciones (que el contrincante anote puntos).

Al recurrir a una política de Atención Visual la complejidad computacional de la implementación aumenta de manera considerable. Se recurrió a un modelo DRAM especializado en enfocar su atención en las áreas más representativas de imágenes histológicas, que aprende mediante Aprendizaje por Refuerzo a predecir de manera adecuada si una imagen histológica presenta tejido canceroso o no. Para dicha implementación, los mejores resultados se obtuvieron con base en los parches de las imágenes de mayor resolución y dimensiones (2560x2560, al nivel 0 de resolución). De igual manera se logró corroborar que el modelo DRAM posee una mayor eficiencia al predecir de manera adecuada glimpses sobre la imagen histológica que representan tejido canceroso cuando se trabaja con imágenes con máscaras detalladas y de menor tamaño a comparación de la totalidad de la imagen. Tiene sentido, debido a que la política de Atención Visual es más eficiente cuando hay que analizar una pequeña porción en específico de un tejido y no la totalidad de este.

## BIBLIOGRAFÍA


ABADI, Martín, *et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (vid. pág. 62).

AKENSERT. *PANDA Optimized Tiling & TF Data Dataset*. `https://www.kaggle.com/code/akensert/panda-optimized-tiling-tf-data-dataset`. Accessed: 2024-07-21. 2022 (vid. pág. 107).

ALVAREZ, waldo. *Example of a Markov Decision Process*. URL: `https://commons.wikimedia.org/wiki/File:Markov_Decision_Process.svg` (vid. pág. 24).

BADIA, Adrià Puigdomènech, *et al.* "Agent57: Outperforming the Atari Human Benchmark". eng. En: *arXiv.org* (2020) (vid. pág. 58).

BAKER, Bowen, *et al.* "Designing Neural Network Architectures using Reinforcement Learning". eng. En: *arXiv.org* (2017) (vid. pág. 52).

BERNARD MARR, Matt Ward. *How 50 Successful Companies Used AI and Machine Learning to Solve Problems*. Wiley, 2019 (vid. pág. 22).

BONACCORSO, Giuseppe. *Machine Learning Algorithms*. Jul. de 2017 (vid. pág. 18).

BRENDAN MARTIN, Satwik Kansal. *Q-Learning Matrix Initialized and After Training*. URL: `https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/` (vid. pág. 27).

BROWNLEE, Jason. *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks*. 2019. URL: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/ (vid. pág. 90).

— *A Gentle Introduction to the Rectified Linear Unit*. 2019. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (vid. pág. 69).

— *Difference Between a Batch and an Epoch in a Neural Network*. 2022. URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20batch%20size%20is%20a%20number%20of%20samples%20processed%20before,samples%20in%20the%20training%20dataset. (vid. pág. 96).

— *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. 2017. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (vid. pág. 81).

— *Understand the Impact of Learning Rate on Neural Network Performance*. 2020. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (vid. págs. 79, 80).

BULTEN, W., *et al.* "Artificial intelligence for diagnosis and Gleason grading of prostate cancer: the PANDA challenge". En: *Nature Medicine* 28 (2022), págs. 154-163. DOI: 10.1038/s41591-021-01620-2 (vid. pág. 104).

CELEBI, M. Emre y AYDIN, Kemal. *Unsupervised Learning Algorithms*. eng. Cham: Springer International Publishing AG, 2016 (vid. pág. 23).

CHEN, Bindi. *Why Rectified Linear Unit (ReLU) in Deep Learning and the best practice to use it with TensorFlow*. 2021. URL: https://towardsdatascience.com/why-

rectified-linear-unit-relu-in-deep-learning-and-the-best-practice-to-use-it-with-tensorflow-e9880933b7ef (vid. pág. 69).

CHEN, Jim X. “The Evolution of Computing: AlphaGo”. eng. En: *Computing in science  engineering* 18.4 (2016), págs. 4-7 (vid. pág. 34).

CHEN, Samuel Yen-Chi, *et al.* “Variational Quantum Circuits for Deep Reinforcement Learning”. En: *IEEE Access* 8 (2020), págs. 141007-141024 (vid. pág. 57).

CHOLLET, Francois, *et al. Keras*. 2015. URL: https://github.com/fchollet/keras (vid. pág. 68).

COLABORATORY. *Welcome To Colaboratory*. 2014. URL: https://colab.research.google.com/notebooks/welcome.ipynb (visitado 2022) (vid. pág. 63).

*Conv2D layer*. URL: https://keras.io/api/layers/convolution_layers/convolution2d/ (vid. pág. 88).

DAHAL, PARAS. *Classification and Loss Evaluation - Softmax and Cross Entropy Loss*. 2018. URL: https://deepnotes.io/softmax-crossentropy (vid. pág. 75).

DOSHI, Sanket. *Various Optimization Algorithms For Training Neural Network*. 2019. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 (vid. pág. 76).

DUTTA, Sayon. *Reinforcement Learning with TensorFlow: A Beginner’s Guide to Designing Self-Learning Systems with TensorFlow and OpenAI Gym*. eng. Birmingham: Packt Publishing, Limited, 2018 (vid. págs. 30, 32).

EDUCATION, IBM Cloud. *What is Gradient Descent?* 2020. URL: https://www.ibm.com/cloud/learn/gradient-descent (vid. pág. 77).

EKKEHARDT ERNST, Rossana Merola y Daniel Samaan. “Economics of Artificial Intelligence: Implications for the Future of Work”. En: *IZA Journal of Labor Policy* 9.Jun (2019) (vid. pág. 22).

ENCORD. *KL Divergence in Machine Learning*. Accessed: 2024-07-21. 2023 (vid. pág. 113).

GANESH, Prakhar. *Types of Convolution Kernels : Simplified*. 2019. URL: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37 (vid. pág. 88).

GMBH, AnyDesk Software. *AnyDesk*. 2012. URL: https://anydesk.com/es (visitado 2021) (vid. pág. 66).

GOODFELLOW, Ian; BENGIO, Yoshua y COURVILLE, Aaron. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (vid. págs. 23, 51).

HARRIS, Charles R., *et al.* “Array programming with NumPy”. En: *Nature* 585.7825 (sep. de 2020), págs. 357-362. DOI: 10.1038/s41586-020-2649-2 (vid. pág. 62).

HAYDARI, Ammar e YILMAZ, Yasin. “Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey”. En: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2022), págs. 11-32. DOI: 10.1109/TITS.2020.3008612 (vid. págs. 19, 32).

HDSP. *Grupo de investigación en diseño de algoritmos y procesamiento de datos multidimensionales*. 2022. URL: http://cormoran.uis.edu.co/eisi/grupo/hdsp (visitado 2021) (vid. pág. 66).

HOLCOMB, Sean D., *et al.* "Overview on DeepMind and Its AlphaGo Zero AI". En: *Proceedings of the 2018 International Conference on Big Data and Education*. ICBDE '18. Honolulu, HI, USA: Association for Computing Machinery, 2018, 67–71. DOI: 10.1145/3206157.3206174 (vid. págs. 54, 55).

HUI, Jonathan. 2018 (vid. pág. 56).

HUNTER, J. D. "Matplotlib: A 2D graphics environment". En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55 (vid. pág. 62).

JAITLEY, Urvashi. *Why Data Normalization is necessary for Machine Learning models*. 2018. URL: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029 (vid. pág. 74).

JETBRAINS. *PyCharm*. 2017. URL: https://www.jetbrains.com/pycharm/ (visitado 2022) (vid. pág. 63).

KOTELUK, Oliwia, *et al.* "How Do Machines Learn? Artificial Intelligence as a New Era in Medicine". En: *Journal of Personalized Medicine* 11.1 (2021). DOI: 10.3390/jpm11010032 (vid. pág. 22).

LI, Yuxi. "Deep Reinforcement Learning: An Overview". eng. En: *arXiv.org* (2017) (vid. pág. 30).

MAHAPATRA, Sambit. *Why Deep Learning over Traditional Machine Learning?* 2018. URL: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063 (vid. pág. 70).

MANEA, Florin; MILLER, Russell G. y NOWOTKA, Dirk, eds. *Sailing Routes in the World of Computation - 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30 - August 3, 2018, Proceedings*. Vol. 10936. Lecture Notes in Computer Science. Springer, 2018. DOI: 10.1007/978-3-319-94418-0 (vid. pág. 54).

MAO, Hongzi, *et al.* "Resource Management with Deep Reinforcement Learning". En: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA: Association for Computing Machinery, 2016, 50–56. DOI: 10.1145/3005745.3005750 (vid. pág. 53).

MIT. *6.S191 Introduction to Deep Learning* (vid. pág. 68).

MITDEEPLEARNING. *MIT 6.S191: Introduction to Deep Learning*. 2017. URL: https://github.com/aamini/introtodeeplearning (visitado 2022) (vid. pág. 62).

MNIH, Volodymyr, *et al.* "Playing Atari with Deep Reinforcement Learning". eng. En: *arXiv.org* (2013) (vid. pág. 58).

MOMENI, Alexandre; THIBAULT, Marc y GEVAERT, Olivier. "Deep Recurrent Attention Models for Histopathological Image Analysis". En: *bioRxiv* (2018). Preprint. DOI: 10.1101/438341 (vid. págs. 59, 107).

NACHUM, Ofir, *et al.* "Bridging the Gap Between Value and Policy Based Reinforcement Learning". eng. En: *arXiv.org* (2017) (vid. pág. 33).

OPENAIGYM. *CartPole-v0*. 2015. URL: https://gym.openai.com/envs/CartPole-v0 (visitado 2021) (vid. pág. 64).

— *Pong-V0*. 2015. URL: https://gym.openai.com/envs/Pong-v0/ (visitado 2021) (vid. pág. 65).

OTTERLO, Martijn y WIERING, Marco. "Reinforcement Learning and Markov Decision Processes". En: *Reinforcement Learning: State of the Art* (ene. de 2012), págs. 3-42. DOI: 10.1007/978-3-642-27645-3_1 (vid. pág. 24).

PASZKE, Adam, *et al.* "PyTorch: An Imperative Style, High-Performance Deep Learning Library". En: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, págs. 8024-8035 (vid. pág. 62).

RAVICHANDIRAN, Sudharsan. *Hands-On Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow*. eng. Birmingham: Packt Publishing, Limited, 2018 (vid. págs. 23, 26).

RODRÍGUEZ, Julián. *Pong Modelo Propuesto primer intervalo*. 2022. URL: https://youtube.com/shorts/o-wspmPKVt4 (vid. pág. 101).

— *Pong Modelo Propuesto último intervalo*. 2022. URL: https://youtube.com/shorts/jVsvRqywv68?feature=share (vid. pág. 102).

— *Pong Modelo Referencia*. 2022. URL: https://www.youtube.com/shorts/gPdclEnfQcU (vid. pág. 99).

— *Péndulo Invertido*. 2022. URL: https://youtu.be/h3AQCP4I0Vg (vid. pág. 84).

RODRÍGUEZ, Julián. *Péndulo Invertido Red Propuesta*. 2022. URL: https://youtu.be/v1qSojOonhs (vid. pág. 85).

SAHA, Sumit. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (vid. pág. 87).

SAHOO, Sabyasachi. *Deciding optimal kernel size for CNN*. 2018. URL: https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363 (vid. pág. 89).

SCIENCE, Towards Data. *How to Understand and Use Jensen-Shannon Divergence*. Accessed: 2024-07-21. 2023 (vid. pág. 113).

SU, Yishan, *et al.* "DQELR: An Adaptive Deep Q-Network-Based Energy- and Latency-Aware Routing Protocol Design for Underwater Acoustic Sensor Networks". eng. En: *IEEE access* 7 (2019), págs. 9091-9104 (vid. pág. 52).

VAN ROSSUM, Guido y DRAKE JR, Fred L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995 (vid. pág. 61).

VERMA, Yugesh. *A Complete Understanding of Dense Layers in Neural Networks*. 2021. URL: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/ (vid. pág. 68).

VOORHEES, E. M. *Mutual Information*. Accessed: 2024-07-21. 2023 (vid. pág. 114).

WATKINS, Christopher J.C.H y DAYAN, Peter. "Technical Note: Q-Learning". eng. En: *Machine learning* 8.3 (1992), pág. 279 (vid. pág. 27).

WUZHT. *Play OpenAI Gym game of Pong using Deep Q-Learning*. 2020. URL: https://github.com/wuzht/DQN_Pong (vid. pág. 91).

XIA, Li. "Mean–variance optimization of discrete time discounted Markov decision processes". eng. En: *Automatica (Oxford)* 88 (2018), págs. 76-82 (vid. págs. 24, 25).

YAMASHITA, Rikiya, *et al.* "Convolutional neural networks: an overview and application in radiology". En: *Insights into Imaging* 9 (2018), págs. 611 -629 (vid. pág. 90).

YOSHIDA, Naoto; UCHIBE, Eiji y DOYA, Kenji. "Reinforcement learning with state-dependent discount factor". eng. En: *2013 IEEE Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. IEEE, 2013, págs. 1-6 (vid. pág. 29).

ZHANG, Aston, *et al. DIVE INTO DEEP LEARNING*. 2020. URL: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html (vid. pág. 88).

ZHANG, Xian-Da. "Machine Learning". En: *A Matrix Algebra Approach to Artificial Intelligence*. Singapore: Springer Singapore, 2020, págs. 223-440. DOI: 10.1007/978-981-15-2770-8_6 (vid. pág. 23).

# ANEXOS

Anexo A. Productos académicos.

- Daniel, R., Julián, R. & David, R. (2024, Julio). IDENTIFYING REGIONS OF CLINICAL INTEREST IN HISTOLOGICAL IMAGES BY DEEP RECURRENT ATTENTION MODELS. Presentado para participar en SIPAIM CONFERENCE.